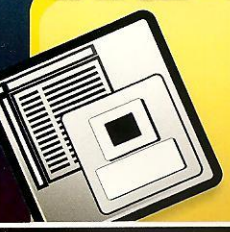
...a edição ...ha acesso ...na WEB a:
...0 vídeo-aulas +150 ...
e 6 cursos online

ClubeDelphi

Ano 7 · Edição 87 · R$9,90 DevMed

Pesquisas sofisticadas com Firebird e Ajax

Na Web



Confira no portal ClubeDelphi PLUS, um curso online completo em **20 partes** abordando todos os recursos do **ClientDataSet!**

ISSN 1517990-7

Boa Idéia
Crie clientes Web em ASP.NET que acessam servidores DataSnap

Mini-curso
Construa um sistema completo de

Mão na Massa
ECF - Saiba como imprimir em impressoras fiscais com Delphi

Segurança
Controle de acesso e níveis de usuários

Performance no Firebird e ADO.NET
DataReaders, DataSets, Connection Poo
Cache e Stored Procedures

Coluna Ask the Expert

Quantas cópias de seu software existem no mercado?

Esta é a chave mais segura do mundo contra pirataria de software.

Comprove você mesmo!

SafeNet®
The Foundation of Information Security

Alameda Tocantins, 280 - Barueri-SP

Tel: +55 11 4208-7700

http://br.safenet-inc.com - Contato: infobrasil@safenet-inc.com

ClubeDelphi

Ano 7 - 87ª Edição - 2007 - ISSN 1517990-7 Impresso no Brasil

Corpo Editorial

Diretor Editorial e Administrativo
Gladstone Matos
gladstone@neoficio.com.br

Editor de Arte
Vinicius O. Andrade
viniciusoandrade@gmail.com

Capa
Antonio Xavier
webdesigner@devmedia.com.br

Editor Geral
Guinther Pauli
guinther@devmedia.com.br

Editor Técnico
Luciano Pimenta
lucianopimenta@clubedelphi.net

Revisão
Rafael de Castro Santos
rafael@devmedia.com.br

Distribuição
Fernando Chinaglia Dist. S/A
Rua Teodoro da Silva, 907
Grajaú - RJ - 206563-900

Atendimento ao Leitor
A DevMedia conta com um departamento exclusivo para o atendimento ao leitor. Se você tiver algum problema no recebimento do seu exemplar ou precisar de algum esclarecimento sobre assinaturas, exemplares anteriores, endereço de bancas de jornal, entre outros, entre em contato com:

Carmelita Mulin
www.devmedia.com.br/central/default.asp
(21) 2220-5375

Kaline Dolabella
Gerente de Marketing e Atendimento
kalined@terra.com.br
(21) 2220-5375

Publicidade
Para informações sobre veiculação de anúncio na revista ou no site entre em contato com:

Kaline Dolabella
publicidade@devmedia.com.br

Para fechar parcerias ou ações específicas de marketing com a DevMedia, entre em contato com:

Jeff Wendell
jeff@devmedia.com.br

ClubeDelphi PLUS

DevMedia GROUP

A revista ClubeDelphi é parte integrante da assinatura ClubeDelphi PLUS. Para mais informações sobre o pacote PLUS, acesse: *http://www.devmedia.com.br/clubedelphi/portal.asp*

Apoio

ASSESPRO PARANÁ

ÍNDICE

06 - AJAX – Buscas incrementais
Fabrício Desbessel

12 - ASP.NET e DataSnap
Guinther Pauli

18 - Performance no Firebird e ADO.NET
Guinther Pauli

24 - Contas a Pagar e Cobrança
Adriano Santos

34 - Impressão em ECF
Adriano Santos

44 - Segurança de Sistemas
Maikel Marcelo Scheid

EDITORIAL

O AJAX representa a segunda geração dos aplicativos para Web, focados na colaboração entre pessoas, um dos pilares da Web 2.0. Não falamos apenas de uma evolução na plataforma Web, mas de uma mudança profunda em como os aplicativos são escritos. Tecnicamente falando, o AJAX permite a comunicação assíncrona de páginas com o servidor Web, permitindo ainda a atualização parcial de pequenas porções da tela, ao invés do refresh total. Suas aplicações vão ser semelhantes a aplicações Desktop e agora sim, cada vez mais veremos sistemas sendo escritos para Web e não mais para Desktop. Nesta edição, o Fabrício traz um excelente artigo que mostra a integração Delphi e AJAX, aliados na construção de uma aplicação de pesquisas incrementais em bancos de dados Firebird.

Nos artigos que produzi para esta edição, o tema principal é ASP.NET. Aprenda a criar clientes em ASP.NET para um servidor DataSnap. Com isso, você poderá integrar seu atual sistema multicamadas com a melhor tecnologia para desenvolvimento Web. No outro artigo, mostro como obter performance no acesso ao banco Firebird usando ADO.NET. Saiba como funciona o Connection Pooling, como usar corretamente DataReaders e DataSets, cache, Stored Procedures e muito mais.

O Adriano inicia nesta edição um mini-curso que mostra como criar um sistema completo de contas a pagar e cobrança, utilizando as principais tecnologias atuais do desenvolvimento Delphi: arquitetura Client/Server, banco Firebird, interface VCL e acesso dbExpress. Em seu outro artigo, Adriano apresenta um artigo que há tempos era solicitado: como imprimir em impressoras matriciais usando o Delphi. E para finalizar, o Maikel fala sobre segurança de sistemas.

Sucesso a todos e boa leitura!

Guinther Pauli
guinther@devmedia.com.br
Microsoft Certified: MCP, MCAD, MCSD.NET
Borland Certified: Delphi 6, 7, 2005, 2006, Web, Kylix

Fale com o Editor

É muito importante para a equipe saber o que você está achando da revista: que tipo de artigo você gostaria de ler, que artigo você mais gostou e qual artigo você menos gostou. Fique a vontade para entrar em contato com os editores e dar a sua sugestão!
Se você estiver interessado em publicar um artigo na revista ou no site ClubeDelphi, entre em contato com os editores, informando o título e mini-resumo do tema que você gostaria de publicar:

Guinther Pauli - Editor da Revista
guinther@devmedia.com.br

Luciano Pimenta - Editor do Site
lucianopimenta@clubedelphi.net

Portal do Assinante

NÃO PERCA

A ClubeDelphi tem uma novidade para você que comprou este exemplar na banca de jornal: você pode acessar GRATUITAMENTE, o Portal do Assinante ClubeDelphi!

Confira o que você encontra no Portal do Assinante:
- Mais de 400 Vídeo Aulas!
- 6 cursos online!
- 1 Livro Eletrônico sobre ADO.NET e BDP!
- Mais de 140 Artigos Exclusivos!

Para Utilizar o Portal do Assinante, acesse *www.devmedia.com.br/clubedelphi/portal.asp* e utilize as informações abaixo: **Login: DVM.CD e Senha: CLB87**

O acesso é válido por 30 dias a partida da data de lançamento da revista. Todos os meses a ClubeDelphi lhe dará uma senha válida para acessar o portal. Comprando a revista regularmente em bancas, você terá acesso ininterrupto a ele!

Ask The Expert

Perguntas e Respostas

Dúvidas respondidas por **Luciano Pimenta**
(envie as suas para pimenta@devmedia.com.br)

Caixa de mensagem com tempo

Estou trabalhando em uma aplicação onde não gostaria que o usuário demorasse a responder confirmações de caixas de mensagens (com os botões Sim ou Não). Tem como eu colocar um timer ou algo que marcasse uma opção default?

Isabel Foster

Primeiro criamos um diálogo utilizando *CreateMessageDialog*. Essa função retorna um objeto formulário com o diálogo, onde podemos adicionar uma barra de progresso. Além disso, adicionamos um *Timer* para atualizar a barra de progresso dinamicamente e exibimos o diálogo utilizando *ShowModal*. Implementamos o evento *OnTimer* do *Timer* para verificar se o tempo limite foi atingido, caso verdadeiro, fechamos o diálogo alterando a propriedade *ModalResult*, pelo código, para *mrCancel*. Caso contrário, utilizamos *StepIt* para atualizar a barra de progresso. Adicione um *Button* a um formulário e adicione o código da **Listagem 1** no *OnClick*:

Adicione na seção *private* o *DialogTimer* e implemente-o com o código mostrado na **Listagem 2**.

Rode a aplicação e veja o resultado (**Figura 1**).

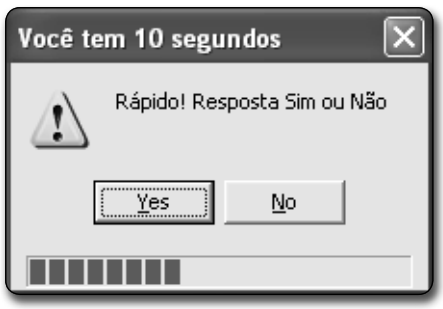


Figura 1. Caixa de mensagem com ProgressBar

Listagem 1. Código evento OnClick do Button

```
uses ExtCtrls, ComCtrls;
...
procedure TForm1.Button1Click(Sender: TObject) ;
var
  AMsgDialog: TForm;
  AProgressBar: TProgressBar;
  ATimer: TTimer;
begin
  AMsgDialog := CreateMessageDialog('Rápido! Resposta Sim ou Não', mtWarning, [mbYes, mbNo]);
  AProgressBar := TProgressBar.Create(AMsgDialog);
  ATimer := TTimer.Create(AMsgDialog);
  with AMsgDialog do

  try
    Tag := 10; //segundos
    Caption := 'Você tem 10 segundos';
    Height := 150;
    with AProgressBar do
    begin
      Name := 'Progress';
      Parent := AMsgDialog;
      Max := AMsgDialog.Tag; //segundos
      Step := 1;
      Top := 100;
      Left := 8;
      Width := AMsgDialog.ClientWidth - 16;
    end;
    with ATimer do
    begin
      Interval := 1000;
      OnTimer := DialogTimer;
    end;
    case ShowModal of
     ID_YES: ShowMessage('Respondeu "Sim".');
     ID_NO: ShowMessage('Respondeu "Não".');
     ID_CANCEL: ShowMessage('Tempo estourado!')
    end;

  finally
    ATimer.OnTimer := nil;
    Free;
  end;

end;
```

Listagem 2. Código de implementação do DialogTimer

```
procedure TForm1.DialogTimer(Sender: TObject) ;
var
  aPB: TProgressBar;
begin
  if NOT (Sender is TTimer) then
    Exit;
  if ((Sender as TTimer).Owner) is TForm then
  with ((Sender as TTimer).Owner) as TForm do
  begin
    aPB := TProgressBar(FindComponent('Progress'));
    if aPB.Position >= aPB.Max then
      ModalResult := mrCancel
    else
      aPB.StepIt;
  end;
end;
```

Assinatura

ClubeDelphi PLUS

Mais conteúdo .NET por menos!

+400 vídeo aulas | 6 cursos online

www.clubedelphi.net/portal

Caro Leitor,

O portal ClubeDelphi PLUS é a continuação, na Web, da revista ClubeDelphi. O portal recebe um conteúdo novo todo dia e hoje conta com: i) mais de 390 vídeo aulas; ii) 6 cursos online; iii) 1 livro eletrônico gratuito, de Guinther Pauli, sobre ADO.NET e BDP; iv) mais de 150 artigos exclusivos (que não foram publicados na revista)!;
Acesse o portal ClubeDelphi PLUS e receba muito mais conteúdo sobre Delphi! E o que é melhor: de graça! Todo leitor da revista ClubeDelphi, seja ele assinante ou comprador da revista em bancas, tem acesso ao portal (para quem compra em bancas, o acesso é válido por 30 dias).
Se você é assinante, utilize o seu login e senha pessoais para acessar o portal. Se você comprou em bancas, utilize o login e senha publicados na página do editorial desta edição.
Confira a seguir as últimas novidades do portal!

Boa leitura e sucesso!
Equipe DevMedia

Curso Online - PLUS

O portal ClubeDelphi PLUS conta com 6 cursos online, confira!

Sistema completo com Delphi 7, dbExpress e Firebird 2.0 [*Luciano Pimenta*]

Aplicações WEB com IW e Delphi 7 (Delphi Win32)! [*Guinther Pauli*]

Aplicações client/server no Delphi 2006 [*Luciano Pimenta*]

Criando uma Aplicação multi-camadas Completa com Delphi [*Guinther Pauli*]

Criando uma aplicação completa: sistema SysCash [*Everson Volaco*]

Aplicações Client/Server com dbExpress e Firebird [*Luciano Pimenta*]

.:: DESTAQUE ::.

Confira no portal ClubeDelphi PLUS, 20 vídeo aulas sobre dicas avançadas de ClientDataSet

Curso Online de ClientDataSet
http://www.devmedia.com.br/articles/listcomp.asp?keyword=cursoclientdataset

Últimas Vídeo-Aulas (Somente para assinantes)

Sistema completo com Delphi 7, dbExpress e Firebird 2.0
Acompanhe a criação de um sistema completo de gerenciamento de locadora, usando Delphi, dbExpress e Firebird 2.0. Luciano Pimenta mostrará todos os passos necessários para a criação do projeto em todas as fases de implementação, desde a criação do banco de dados, até a migração para .NET.

Criando uma tela de consulta dinâmica – Partes IV e V
Veja nessa vídeo aula de Paulo Quicoli, como criar um tela de consulta dinâmica.

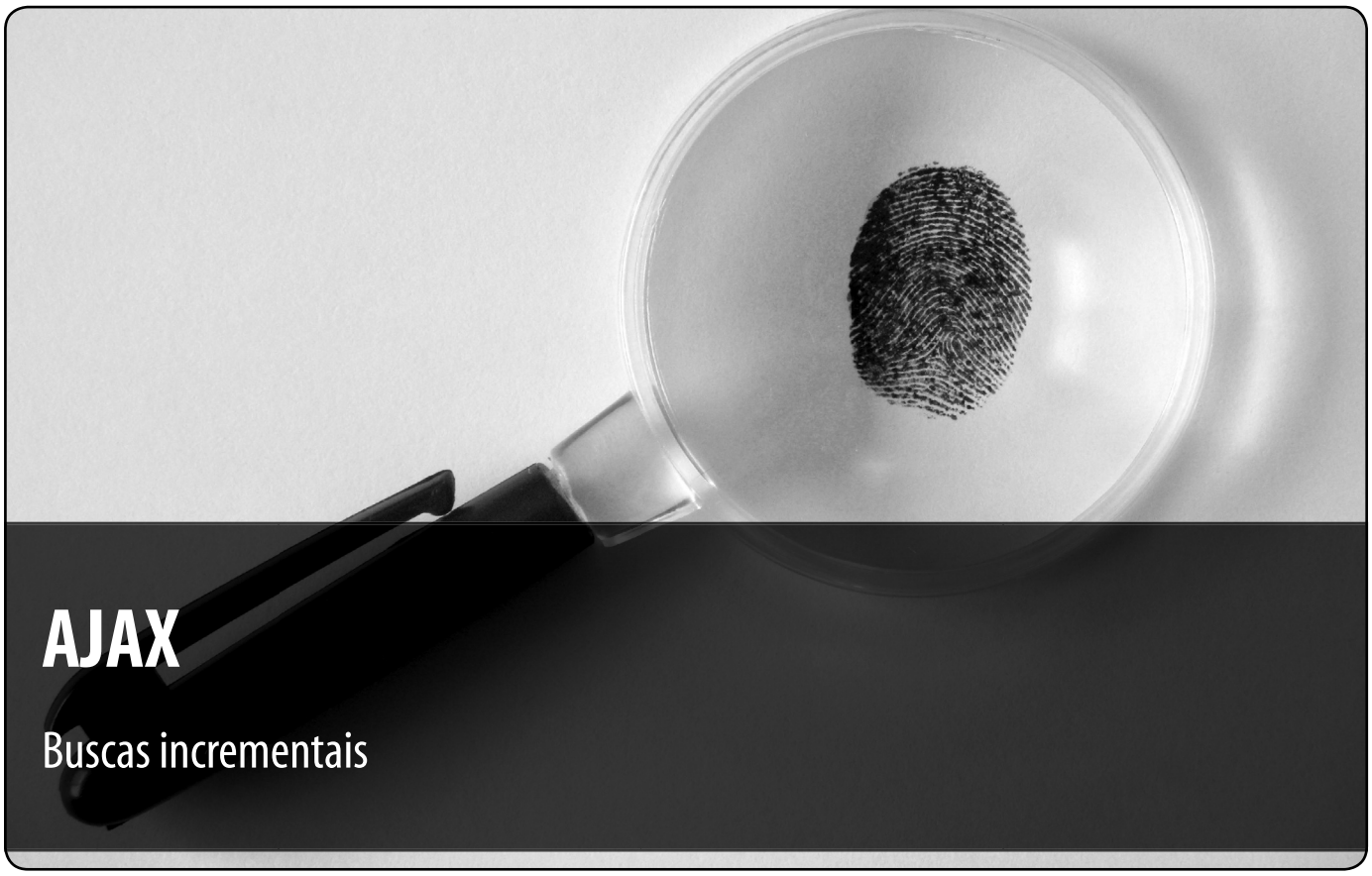
Mini-curso de ClientDataSet - Partes VII a XVI
Veja nessas aulas de Guinther Pauli um mini-curso exclusivo sobre dicas avançadas de ClientDataSet.

Componente Window (Janela) no Delphi for PHP
Veja nessa vídeo aula de Fabricio Desbessel, como trabalhar com o componente Window no Delphi for PHP.

Apresentando a suíte UserControls - Parte II
Veja nessa vídeo aula de Paulo Quicoli a suíte User Controls.

Conhecendo o plug-in Castalia
Veja nessa vídeo aula de Adriano Santos, como utilizar o plugin castalia.

Plug-in CanTools
Veja nessa vídeo aula de Adriano Santos, como utilizar o plugin CanTools.



AJAX

Buscas incrementais

Fabrício Desbessel
(fabricio@fabricio.pro.br)
é professor de Linguagem de Programação do Curso Técnico em Informática do Colégio Frederico Jorge Logemann de Horizontina/RS e da FAHOR Faculdade Horizontina. Delphiano de coração está sempre disposto a provar que com o Delphi sempre teremos a melhor solução. Site www.fabricio.pro.br.

A tecnologia AJAX continua sendo a mais importante inovação em aplicações Web. Temos vários artigos e vídeo aulas que mostram a utilização dessa tecnologia em inúmeros ambientes e frameworks. Na edição 75 apresentei o MagicAjax, um framework que agiliza em muito o desenvolvimento no Delphi 2006 de aplicações com a tecnologia AJAX.

Inúmeros foram os e-mails que recebi com questões de como e onde utilizar o AJAX. Um questionamento interessante foi verificar a possibilidade de fazer uma busca incremental utilizando o AJAX, ao melhor estilo do Google, ou seja, você digitará as iniciais do texto e os resultados apresentados serão exibidos através de uma consulta ao banco de dados.

Isso tudo sem *refresh* na página da pesquisa. Criei um exemplo e mostro neste artigo como implementá-lo, passo a passo.

MagicAjax

Esse framework é gratuito, está na *release 0.3.0* e pode ser baixado no endereço oficial do projeto: *www.magicajax.net*. Para utilizar, precisamos adicionar uma referência para a DLL do MagicAjax no projeto ASP.NET. Crie um novo projeto do tipo *ASP.NET Web Application - Delphi for .NET*. No *Project Manager* selecione o item *References* e clique com o botão direito para abrir o menu de contexto e escolha *Add Reference*.

Na tela de *Add Reference*, com a aba *.NET Assemblies* selecionada, clique no *Browse* e encontre o arquivo *MagicAjax.dll* a partir do diretório onde você o descompactou. Clique em OK para fechar o editor (**Figura 1**).

No *Project Manager*, clique duas vezes no arquivo *Web.config* pois precisaremos editá-lo, informando à aplicação para tratar as requisições de forma diferenciada, através do *MagicAjax*.

Procure pela sessão <*httpModules*> e

adicione o seguinte código:

```
<add name="MagicAjax"
 type="MagicAjax.MagicAjaxModule, MagicAjax"/>
```

Também podemos adicionar os componentes à *Tool Palette*, através do menu *Component>Install .NET Components*. Clique no *Select an Assembly*, encontre a DLL e clique em OK. Agora teremos disponível os componentes da **Figura 2**.

Pesquisa incremental

Vamos criar uma consulta de dados na tabela *Employee* do banco *Employee.fdb*, através do *Firebird Data Provider* (veja box). Adicione um *FbConnection*, clique na propriedade *ConnectionString* e configure suas propriedades conforme a **Figura 3**.

Agora, adicione um *FbCommand*, configurando sua propriedade *Connection* para o *fbConnection1* e *CommandText* conforme o código a seguir:

```
select EMP_NO, FIRST_NAME
from EMPLOYEE
where Upper(FIRST_NAME) like @NOME
order by FIRST_NAME
```

Abra o editor da propriedade *Parameters* do *FbCommand* e adicione um novo parâmetro, configurando seu *ParameterName* para "@NOME" e *SourceColumn* para "FIRST_NAME". Verifique se o *FbDbType* está como *VarChar*.

Design da aplicação

Inicialmente devemos colocar o *AjaxPanel* encontrado na paleta *Ajax*. Aumente seu tamanho, pois todos os outros componentes serão dispostos dentro do mesmo. O *AjaxPanel* funciona como um painel de componentes e todas as ações dos componentes que estão dentro serão realizada através de chamadas assíncronas ao servidor, ou seja, utilizando a tecnologia AJAX.

Dentro do *AjaxPanel*, digite o texto "Primeiro Nome:" e, abaixo, insira um *TextBox* da paleta *Web Controls*. Ao lado insira um outro *TextBox* e reduza seu tamanho. Abaixo do *TextBox1* acrescente um *ListBox*.

Uma pesquisa incremental tem seu funcionamento a cada tecla que o usuário digita, trazendo resultados referentes às iniciais digitadas e possibilitando a escolha de um dos resultados que completará o texto.

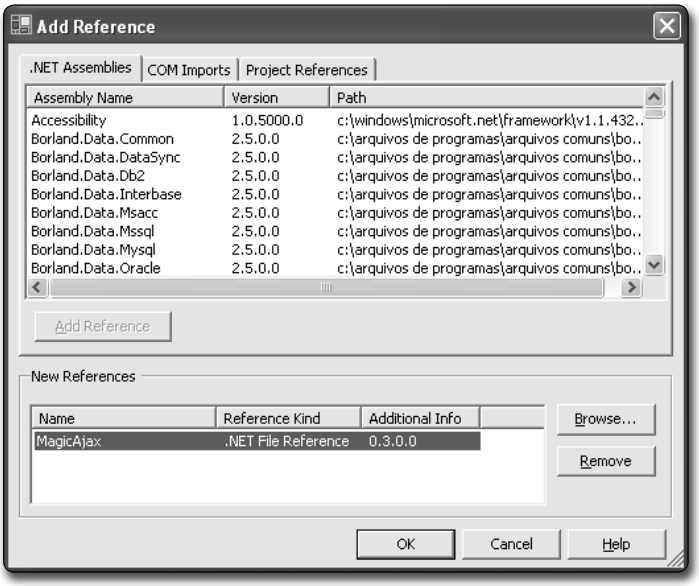


Figura 1. Adicionando referência ao MagicAjax

Figura 2. Componentes instalados no Delphi

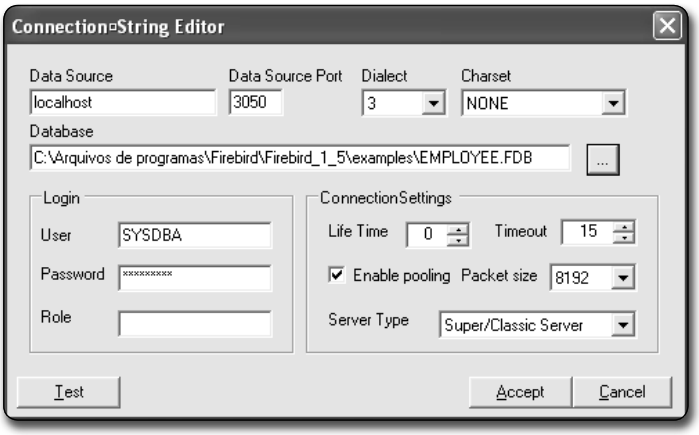


Figura 3. Propriedades de conexão

Então, no *TextBox1* colocaremos o código da pesquisa ao banco de dados, no evento *TextChanged*, que corresponde a alteração do texto, conforme **Listagem 1**.

Funcionalidades da aplicação

Para o evento *TextChanged* ser disparado é necessário alterar a propriedade *AutoPostBack* para *True*. Pode parecer estranho habilitar o *postback*, mas fique tranqüilo, pois o *MagicAjax* interceptará e fará uma chamada assíncrona. Como essa é a única forma de fazer uma pesquisa incremental fica claro que sem AJAX nunca pensaríamos em fazer esse tipo de pesquisa em uma aplicação Web, pois a cada alteração de texto, teríamos que esperar a página "ir e voltar", levando um bom tempo.

Você já pode fazer o primeiro teste, porém você deverá digitar uma letra inicial, por exemplo, "a" e pressionar o TAB para que aconteça o *postback*, ou seja, disparar o evento no servidor.

Firebird Data Provider

Para acessar o Firebird a partir de aplicações ASP.NET, a melhor opção é usar o provider do banco para o ADO.NET, chamado Firebird Data Provider. Para instalá-lo, acesse o endereço www.firebirdsql.com, clique no link Download e a seguir em Firebird .NET Data Provider. Baixe e instale a última versão do Data Provider for .NET Framework 1.1, bastando seguir os passos no assistente que será iniciado.

Neste artigo usaremos versão 1.7 do provider, versão mais recente disponível até o fechamento desta edição. Após a instalação, no Delphi 2005/2006 clique no menu Component|Installed .NET Components. Digite "Firebird Data Provider" na opção Category, clique em Select an Assembly e escolha o arquivo FirebirdSql.Data.Firebird.dll, localizado no diretório de instalação do provider, por padrão em C:\Arquivos de programas\FirebirdNETProvider(versão).

Clique em OK e observe que os novos componentes para acesso ao Firebird estão disponíveis no IDE. Maiores informações sobre o funcionamento e utilização do Firebird Data Provider podem ser encontradas na edição 66 da ClubeDelphi, ou ainda no curso de Delphi e ASP.NET da DevMedia (www.devmedia.com.br/curso/ecommerce2005).

Como queríamos que a cada tecla a pesquisa fosse realizada teremos que utilizar alguns códigos *JavaScript* para controlar melhor a aplicação. No *Page_Load* da página e coloque o seguinte código:

```
{ Fazer postback a cada digitação ou retorno }
TextBox1.Attributes.Add('onkeyup',
  '__doPostBack(''TextBox1'','''')');
```

Utilizando o *Attributes* podemos adicionar eventos e códigos *JavaScript* em um componente. O que fizemos no *TextBox1* foi chamar o *postback* a cada vez que uma tecla for pressionada. Teste novamente a aplicação e informe duas letras, por exemplo, "a" e "s" e veja que a pesquisa será mais próxima do resultado pretendido. Também já é possível retornar com o *backspace* e a pesquisa ser refeita (**Figura 4**).

Agora vamos cuidar de ocultar o *ListBox1* e mostrar quando o resultado da pesquisa tiver itens para exibir. Clique no componente e mude sua propriedade *Visible* para *False*. Agora altere o código do *TextChanged* do *TextBox1* conforme mostra a **Listagem 2**.

No código da **Listagem 2** declaramos uma variável de tipo *StringBuilder* com a finalidade de criar o código *JavaScript*. Verificamos se a consulta ao banco de dados retornou algum resultado, através da propriedade *Items.Count* do *ListBox* e adicionamos um código *JavaScript* para tornar visível ou não.

No final do código, verificamos se já não existe registro do código *JavaScript* na página e registramos, fazendo com que esse código seja executado. Para fazer o auto-completar que, ao selecionar um item no *ListBox* com um clique de mouse, seja preenchido os campos, coloque o código a seguir no evento *SelectedIndexChanged* e altere a propriedade *AutoPostBack* para *True*:

```
TextBox1.Text := ListBox1.SelectedItem.Text;
TextBox2.Text := ListBox1.SelectedValue;
```

Ainda pensando na usabilidade, faremos com que seja possível ao usuário digitar as iniciais e se aparecerem itens no *ListBox* ele possa navegar nesses itens com as setas direcionais (para cima e para baixo).

Para tanto utilizaremos novamente

Listagem 1. Código do TextChanged do TextBox1

```
FbConnection1.Open;
try
  FbCommand1.Parameters['@NOME'].Value:=TextBox1.Text.ToUpper +'%';
  ListBox1.DataSource := FbCommand1.ExecuteReader;
  ListBox1.DataTextField := 'FIRST_NAME';
  ListBox1.DataValueField := 'EMP_NO';
  ListBox1.DataBind;
finally
  FbConnection1.Close;
end;
```

Listagem 2. TextChanged do TextBox1 exibindo/ocultando o ListBox

```
uses System.Text;
...
procedure TWebForm1.TextBox1_TextChanged(sender: System.Object; e: System.EventArgs);
var
  Codigo: StringBuilder;
begin
  fbConnection1.Open;
  try
    FbCommand1.Parameters['@NOME'].Value := TextBox1.Text.ToUpper+'%';
    ListBox1.DataSource := FbCommand1.ExecuteReader;
    ListBox1.DataTextField := 'FIRST_NAME';
    ListBox1.DataValueField := 'EMP_NO';
    ListBox1.DataBind;
    Codigo := StringBuilder.Create;
    Codigo.Append('<script language="JavaScript">)');
    if ListBox1.Items.Count > 0 then
    begin
      Codigo.Append('document.getElementById( '+
        '"ListBox1$ajaxdest").style.visibility = '+'"visible"');
      ListBox1.Visible := True;
    end
    else
      Codigo.Append('document.getElementById('+
        '"ListBox1$ajaxdest").style.visibility = '+'"hidden"');
      Codigo.Append('</script>)');
      if not IsClientScriptBlockRegistered('mostra') then
        RegisterClientScriptBlock('mostra', Codigo.ToString);
  finally
    FbConnection1.Close;
  end;
end;
```

funções *JavaScript* para verificar se a tecla pressionada for a seta para baixo (código 40) e mudaremos o foco para o *ListBox1* selecionando a primeiro item.

No evento *Page_Load* da página substitua o código que colocamos anteriormente pelo código da **Listagem 3**.

No código anterior, acrescentamos códigos *JavaScript* para dois eventos que podem acontecer: *KeyPress* e *KeyUp*. O *KeyPress* acontece antes que o *KeyUp* e por isso nesse evento tornamos o *ListBox* visível. No *KeyUp* verificamos a tecla pressionada e se for a seta para baixo mudamos o foco para o *ListBox* e selecionamos o primeiro item.

Para fazer com que, ao pressionar o ENTER no *ListBox*, os campos sejam completados com o item selecionado e o *ListBox* seja escondido, no *Page_Load* acrescente o código da **Listagem 4**.

Com isso, você já pode testar a aplicação com todo seu funcionamento. A aplicação deve ser representada conforme a **Figura 5**.

Para deixar a aplicação mais transparen-

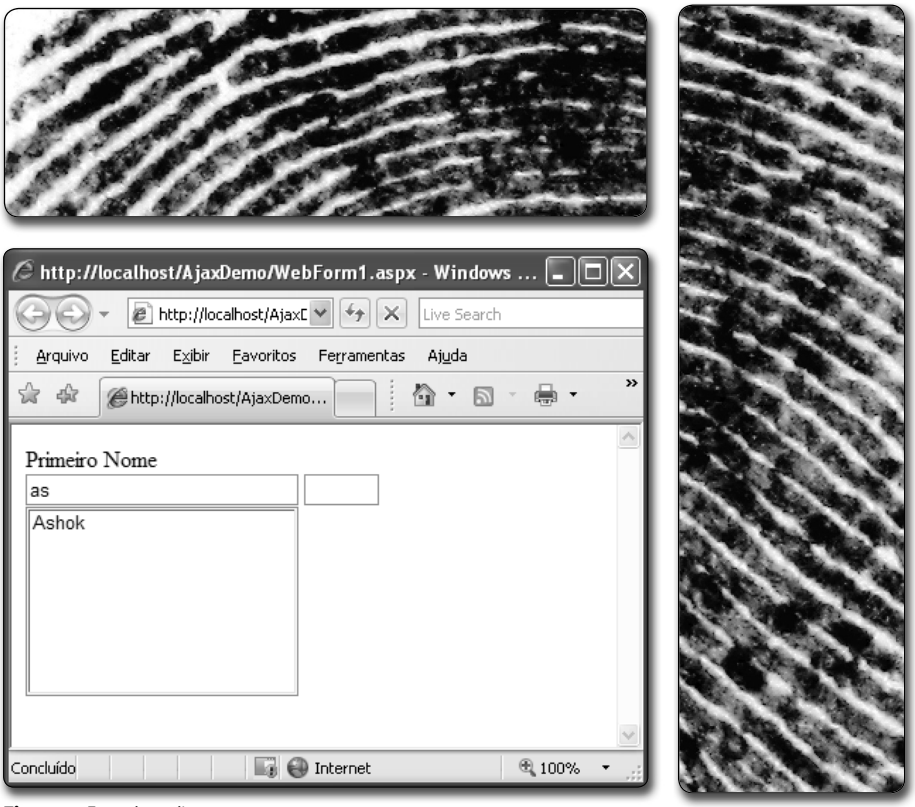
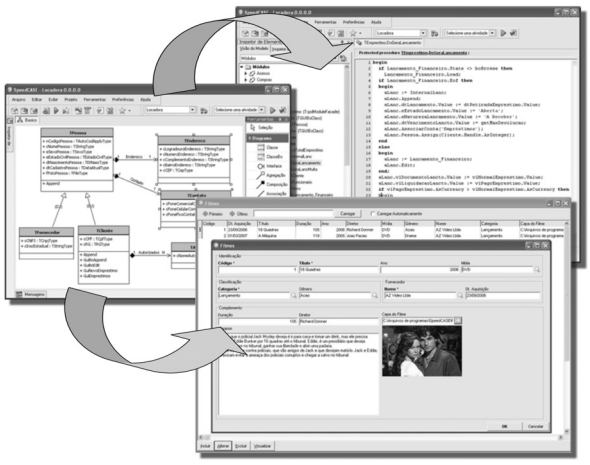


Figura 4. Testando a aplicação

Reduza o tempo de desenvolvimento em até 80%



Speed CASE™

Ambiente de Análise e Implementação Integrados.
Interface Gráfica e Banco de Dados gerados a partir do modelo.

Ferramenta de produtividade para desenvolvimento de software

Conheça a versão Free!

Persistência com:
Interbase
Firebird
MySQL
MS SQL Server

Compatível com:
Turbo Delphi
Delphi
Kylix

Desenvolva para:
Windows
Linux

Faça do tempo seu diferencial

TecnoSpeed TI - Fone: (44) 3028-3749 - **www.speedcase.com.br**

te, vamos retirar a mensagem de *Loading* que o MagicAjax mostra, enquanto envia e retorna informações ao servidor. Para fazer isso, teremos que modificar os arquivos *JavaScript* do MagicAjax. Abra o arquivo *Web.config* e abaixo de *<configuration>* coloque a declaração da **Listagem 5**.

Crie uma pasta com o nome "Script" no diretório de sua aplicação e copie os arquivos *AjaxCallObject.js* e *WebParts.js* para essa pasta. Esses arquivos são encontrados dentro da pasta *Core\Script*, onde foi descompactado o MagicAjax.

O MagicAjax cria esses arquivos automaticamente quando a aplicação é executada. Estamos fazendo isso, pois queremos personalizar a mensagem exibida. Podemos alterar o texto, mudar a cor ou simplesmente fazer com que a mesma não seja exibida.

Neste exemplo, vamos fazer com que a mensagem não seja mais exibida. Abra o arquivo *AjaxCallObject.js* e comente as linhas do *CreateWaitElement()*; colocando "//" (duas barras).

Você encontrará duas chamadas para a função. Depois de comentar teste novamente a aplicação para verificar que a mensagem não será mais exibida.

Conclusão

Com a chegada da tecnologia AJAX não poderemos mais utilizar a desculpa que aplicações Web são diferentes de desktop. Com certeza pode ser mais difícil para implementarmos as facilidades para o usuário.

Mas aí que vem a grande pergunta: Você quer facilidade para você que é desenvolvedor ou quer facilidade para o usuário do seu sistema? Quanto mais concorrência, mais devemos pensar em nossos clientes.

Coloquem a mão na massa e não deixem de utilizar AJAX em suas aplicações Web. ●

Listagem 3. Código do Page_Load alterado

```
{ Verifica se a tecla pressionada for seta para baixo(40) Se for: Verifica se ListBox está
visível. Muda o foco para o ListBox. Seleciona o primeiro Item e se não for: Executa um
PostBack }

TextBox1.Attributes.Add('onkeyup',
  'if((event.keyCode==40) && '+
  '(document.getElementById(''ListBox1$ajaxdest'').'+
  'style.visibility==''visible''))(document.'+
  'getElementById(''ListBox1'').focus();'+
  'document.getElementById(''ListBox1'').'+
  'selectedIndex=0; } else { __doPostBack'+
  '(''TextBox1'','''');}');
{ Torna o ListBox visível }
TextBox1.Attributes.Add('onkeypress',
  'document.getElementById'+
  '(''ListBox1$ajaxdest'').style.'+
  'visibility=''visible''');
```

Listagem 4. Sumindo o ListBox ao pressionar ENTER

```
...
{ Verifica se a tecla pressionada for ENTER (13) Se for, torna o ListBox oculto }
ListBox1.Attributes.Add('onkeypress',
  'if(event.keyCode==13){document.'+
  'getElementById(''ListBox1$ajaxdest'').'+
  'style.visibility=''hidden''}');
```

Listagem 5. Declaração de arquivos JavaScript do MagicAjax

```
<configSections>
  <section name="magicAjax" type="MagicAjax.Configuration.MagicAjaxSectionHandler,MagicAjax"/>
</configSections>
<magicAjax scriptPath="~/Script">
  <pageStore/>
</magicAjax>
```

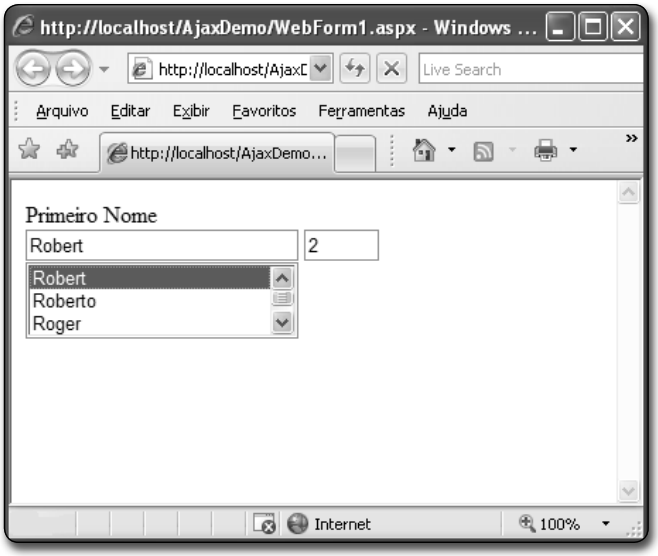


Figura 5. Aplicação em funcionamento

ClubeDelphi PLUS www.devmedia.com.br/clubedelphi/portal.asp

Acesse agora o mesmo o portal do assinante ClubeDelphi e assista a uma série de vídeo aulas de Fabricio Desbessel, sobre AJAX no Delphi.

www.devmedia.com.br/articles/listcomp.asp?txtsearch=utiliza%E7%E3o+da+Tecnologia+Ajax

Unimos a inovação e excelência da **Alog** com
o pioneirismo da **.comDomínio** para criar a
Alog Data Centers do Brasil:

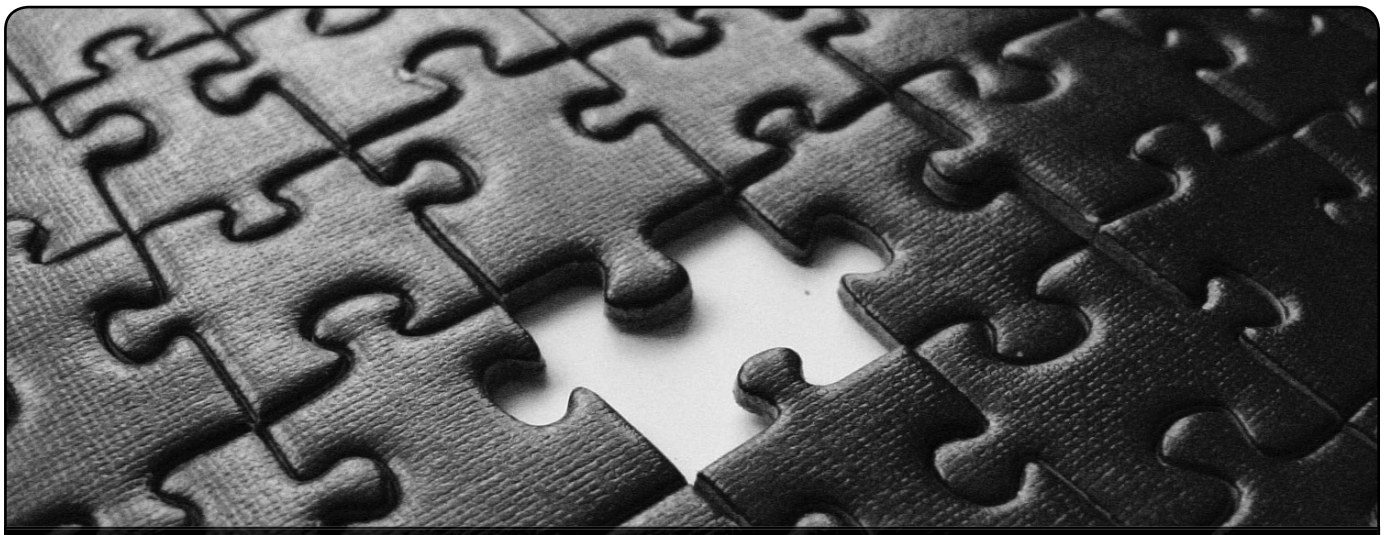
LÍDER EM HOSTING GERENCIADO®

CONHEÇA OS NÚMEROS DESTA LIDERANÇA

2 Data Centers de Classe Mundial
200 Profissionais Dedicados
800 Clientes Corporativos
3.000 Servidores Hospedados
8.000 m² de Área Construída

www.alog.com.br | 0800 282 3330

Hosting Gerenciado® | Colocation | Contingência | Email Corporativo | Conectividade | Serviços Profissionais



ASP.NET e DataSnap

Criando clientes Web para servidores Midas

Guinther Pauli
(guinther@devmedia.com.br)
é autor de mais de 100 artigos publicados e 200 vídeo-aulas e do livro "Delphi – Programação para Banco de Dados e Web".É Bacharel em Sistemas de Informação, Microsoft Certified: MCP, MCAD, MCSD.NET, Borland Certified: Delphi 6, 7, 2005, 2006, Web e Kylix. Editor Geral das Revistas .net Magazine, ClubeDelphi, WebMobile (.NET) e Mr.Bool. Palestrante em vários eventos pelo Brasil, como TechDay SP,RJ,POA,BH, Web Days e todas as edições da Borland Conference.

O DataSnap tem sido amplamente utilizado para a criação de aplicações distribuídas. O Delphi oferece um mecanismo quase que imbatível em produtividade nesse sentido. Podemos criar aplicações multicamadas de uma forma RAD e simples, sem nos preocuparmos com detalhes internos à tecnologia, como protocolos de comunicação.

Muitos desenvolvedores gastaram horas, dias, meses, construindo um servidor de aplicação robusto que concentre todo o acesso a dados e regras de negócio em uma única camada, compartilhada por todas as aplicações clientes.

Quando falamos em DataSnap, no entanto, estamos falando em VCL. Muitos sem dúvida sentiram a necessidade de oferecer não só um cliente desktop parar o servidor de aplicação, mas uma interface Web, de forma que clientes pudessem acessar a solução multicamadas de qualquer local do planeta, usufruindo de todos os recursos e códigos implementados no servidor de aplicação.

O Delphi for .NET deu suporte a uma poderosa tecnologia para o desenvolvimento de aplicações Web: o ASP.NET. O ASP.NET é sem dúvida hoje a tecnologia mais robusta para o desenvolvimento de soluções Web, e muitos desenvolvedores Delphi começaram a utilizá-la para novas soluções.

Mas como ficam as aplicações multicamadas existentes, onde todo o BD, acesso a dados e regra de negócio já estão implementados? Não seria interessante oferecer uma opção Web para a mesma arquitetura, convivendo em harmonia com aplicações cliente desktop já existentes?

Esse é o propósito deste artigo: mostrar como construir uma interface Web para um servidor DataSnap, usando a tecnologia mais robusta para esse propósito, o ASP.NET. Clientes desktop Win32 e Web poderão conviver lado a lado e poderão

compartilhar a mesma camada de negócio e acesso a dados, ou seja, o mesmo servidor de aplicação.

Preparando o banco de dados

Neste exemplo vou utilizar um banco de dados no Firebird 2.0, mas sinta-se a vontade para utilizar o banco de dados de sua preferência. Para facilitar, vamos criar uma tabela com uma estrutura bastante simplificada dentro de um novo banco de dados, chamado "CUSTOMER.FDB".

Para criar o banco e a tabela "CUSTOMERS" você pode utilizar o IBExpert (*www.ibexpert.com*). A estrutura da tabela é mostrada na **Figura 1.** O *script* de criação da tabela pode ser visto na **Listagem 1**.

Ao criar a tabela, veja que definimos um *Sequence* para incrementar automaticamente o campo chave, nesse caso *CustomerId*. Se estiver usando o modo de designer do IBExpert para criar a tabela, basta marcar a opção *AutoInc* e marcar a seguir os itens *Create Generator/Sequence* e *Create Trigger*.

Criada a tabela, adicione alguns registros nela para facilitar nossos testes quando formos criar a aplicação no Delphi.

Criando o servidor DataSnap

No Delphi, clique em *File>New>Other>Multitier>Transactional DataModule*. Na caixa de diálogo que aparece, digite "DM" para o nome da *CoClass*. Uma *CoClass* é na verdade uma classe que implementa uma interface COM. Salve a unit com o nome de "uDM.pas" e o projeto com o nome de "ServerComPlus.dpr", dentro de um diretório de mesmo nome.

Usando componentes dbExpress, configure uma conexão para o banco de dados criado, através do editor de conexões do *SQLConnection*. Lembre-se de colocar *localhost* na frente do caminho do banco para que seja usado TCP/IP e aplicações Web possam acessar o banco (elas são *multi-thread*, conexões locais não funcionam). Configure o *LoginPrompt* para *False*.

Coloque um *SQLDataset* ("sqlCustomers") e aponte-o para a conexão. Na propriedade *CommandText* retorne os registros da tabela CUSTOMERS conforme o *select* da **Listagem 2**.

Coloque um *DataSetProvider* ("dspCustomers") apontando para esse *SQLDataSet*. Configure o parâmetro do *SQLDataSet* para *Input* e *ftString*. Dê também um valor padrão de "%" para ele.

Dê um *Build* no projeto. Agora vamos instalar o servidor no catálogo do COM+. Para isso, podemos usar o próprio IDE do Delphi, através do menu *Run>Install COM+ Objects*.

Na caixa que aparece, marque o objeto DM e na outra caixa, na opção *Install into a new Application* digite "ServerComPlus". Clique OK e OK. Pronto, nosso servidor está instalado e pronto para ser acessado.

Requisitos para acessar o COM+ a partir do ASP.NET

Três etapas precisam ser feitas para podermos acessar o servidor COM+ DataSnap a partir de aplicações ASP.NET. A primeira diz respeito à segurança do aplicativo, a segunda é relativa à compatibilidade dos aplicativos ASP.NET com o COM+ e a terceira diz respeito a interoperabilidade entre .NET e Win32, mais especificamente ADO.NET e DataSnap / ClientDataSet.

Vamos ao primeiro passo. Toda vez que um cliente instancia um servidor COM+, por padrão o objeto assume a identidade do usuário interativo, ou seja, do usuário logado no Windows.

Quando acessamos uma página Web, o ASP.NET juntamente com o IIS (servidor Web) utiliza um usuário padrão para instanciar o aplicativo, chamado *IUSR_NomeDaMaquina*, a menos que a autenticação esteja configurada no aplicativo para usar algo diferente.

O problema é que esse usuário, que será automaticamente usado pela nossa aplicação ASP.NET, não possui privilégios para instanciar objetos COM. Nesse caso temos várias soluções. Uma delas seria

ClubeDelphi PLUS www.devmedia.com.br/clubedelphi/portal.asp

Acesse agora o mesmo o portal do assinante ClubeDelphi e assista a uma vídeo aula de Guinther Pauli que mostra como criar servidores COM+.

www.devmedia.com.br/articles/viewcomp.asp?comp=983

Fields | Description

COMPANY VARCHAR(50) NOT NULL

#	PK	Field Name	Field Type	Size	Not Null	AutoInc
	1	CUSTOMERID	INTEGER		☒	☒
		FIRSTNAME	VARCHAR	50	☒	☐
		COMPANY	VARCHAR	50	☒	☐

Figura 1. Estrutura da tabela CUSTOMERS

Listagem 1. Código de criação da tabela CUSTOMERS

```
CREATE SEQUENCE GEN_CUSTOMERS_ID;

CREATE TABLE CUSTOMERS (
    CUSTOMERID  INTEGER NOT NULL,
    FIRSTNAME   VARCHAR(50) NOT NULL,
    COMPANY     VARCHAR(50) NOT NULL
);

ALTER TABLE CUSTOMERS ADD CONSTRAINT
PK_CUSTOMERS PRIMARY KEY (CUSTOMERID);

CREATE TRIGGER CUSTOMERS_BI FOR CUSTOMERS
ACTIVE BEFORE INSERT POSITION 0
AS
BEGIN
  IF (NEW.CUSTOMERID IS NULL) THEN
    NEW.CUSTOMERID = GEN_ID(GEN_CUSTOMERS_ID,1);
END
```

Listagem 2. Código SQL do SQLDataSet

```
select
  CUSTOMERID,
  FIRSTNAME,
  COMPANY
from
   CUSTOMERS
where
  FIRSTNAME like :FIRSTNAME
```

trocar a conta para acesso anônimo, o que não seria uma boa idéia. Outra seria configurar a identidade do aplicativo ASP.NET, usando o *impersonate*.

A que escolhi e acredito ser a mais segura e fácil de implementar foi dizer ao COM+ para rodar o objeto servidor com uma conta de um usuário específico. Para fazer isso, acesse os *Serviços de Componentes* dentro do *Painel de Controle>Ferramentas Administrativas*. O que você vai ver é uma interface que permite configurar várias opções dos objetos registrados, como pacotes, componentes etc.

Localize nossa aplicação, que deve se chamar *ServerComPlus*, e dê um clique de direita para abrir as propriedades. Na aba *Identity* configure as opções como mostrado na **Figura 2**, indicando uma conta válida (pode ser o seu usuário) na opção *This User*. A seguir, desmarque a opção mostrada na **Figura 3.**

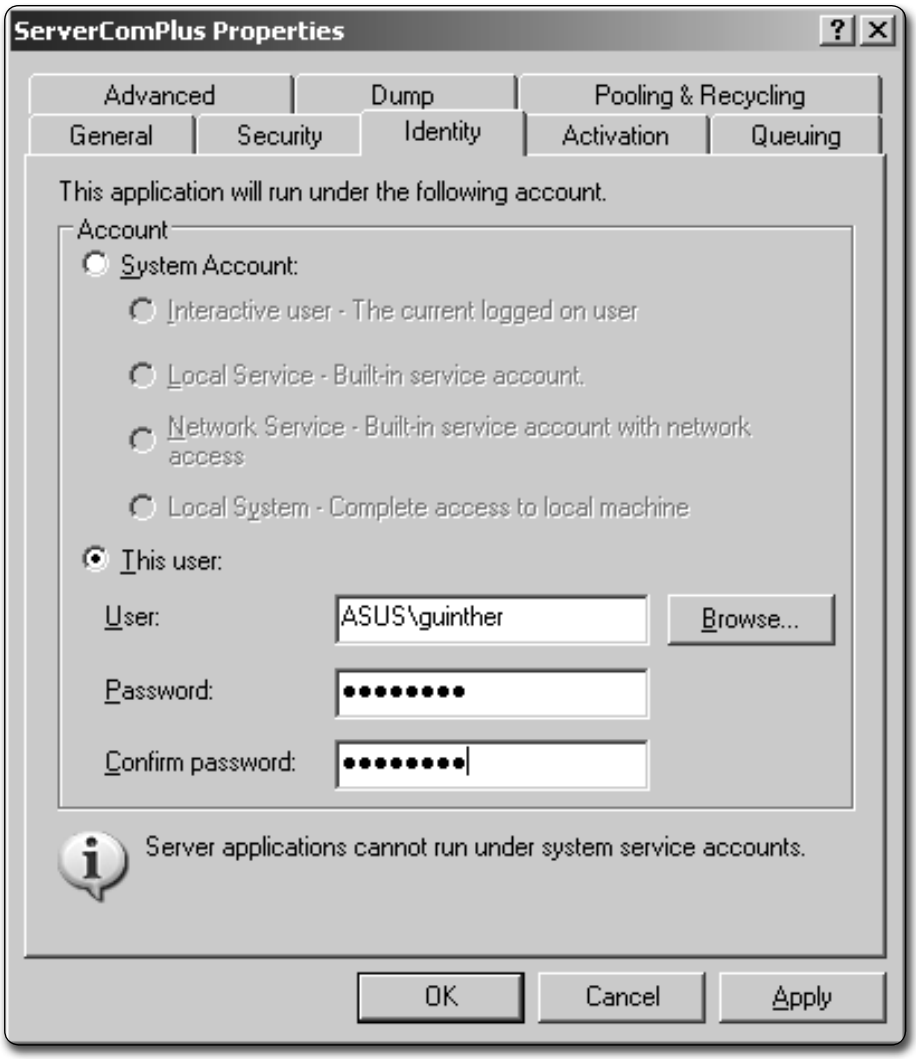


Figura 2. Configurando a conta do usuário COM+

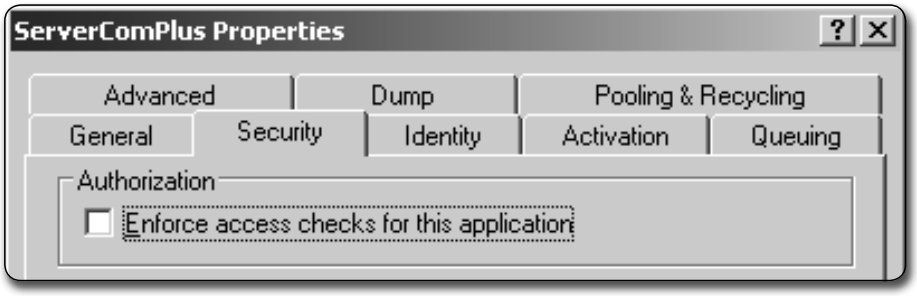


Figura 3. Desmarcando o Enforce Access Checks

A segunda modificação é na aplicação ASP.NET. Você deve modificar a página que vai acessar o aplicativo COM+, acessando o código-fonte do arquivo ASPX. A diretiva *Page* deve conter agora o atributo *AspCompat:*

```
<%@ Page language="c#" Debug="true"
  Codebehind="WebForm1.pas"
AutoEventWireup="false"
  Inherits="WebForm1.TWebForm1"
AspCompat="true" %>
```

Essa configuração permite que a página ASP.NET utilize o mesmo modelo de *thread* utilizado pelo servidor COM+, nesse caso por padrão é *single thread apartment* (STA).

O terceiro passo, e mais complicado, é prover a comunicação entre ASP.NET e DataSnap. O grande problema é que um aplicativo ASP.NET não vai reconhecer os formatos de *DataPacket* enviados por um servidor DataSnap.

Uma solução seria usar XML, mas isso envolveria criar um servidor SOAP e não COM+. Veremos como resolver esse problema a seguir.

> **Nota**: Antes de prosseguir, eu realmente sugiro que você crie uma aplicação cliente para teste em Win32, para ver se está tudo funcionando no seu servidor DataSnap COM+. Isso porque a maioria das mensagens de erro que possam vir a acontecer na aplicação ASP.NET, devido a um problema no servidor, não descreverão a causa exata do problema.

Criando a aplicação cliente ASP.NET

Vamos partir para o último passo, que é a criação da aplicação cliente. Usando Delphi for .NET inicie uma nova aplicação ASP.NET. Já na página principal, no arquivo ASPX, faça a alteração na diretiva *AspCompat* como descrito anteriormente. Desenhe um formulário como mostra a **Figura 4**.

Os nomes dados aos componentes são: "btSelect" e "btApply" para os botões, "tbFind" para o *TextBox* e "DG" para o *DataGrid*. Para colocar o botão *Edit* no Grid, basta selecionar o DG e no *Object Inspector* acessar a opção *Property Builder*. No editor, acesse *Columns* e em *Button Column* adicione uma coluna do tipo *Edit, Update e Cancel*.

Agora precisamos obter os dados do servidor DataSnap. Para isso, no *Click* do *Select* digite o código da **Listagem 3**. O código do *GridDataBind* é mostrado na **Listagem 4**.

O método da **Listagem 3** mostra um personagem novo na história. Para obter os dados do servidor DataSnap, utilizamos um *DataSet* do ADO.NET para colocar os dados em memória (sessão). Porém, o formato de dados desse *DataSet* não é o mesmo que é devolvido por um *IAppServer* do servidor DataSnap. O formato usado pelo *ClientDataSet* em aplicações multicamadas se chama *DataPacket*, e não é compatível com o *DataSet* do ADO.NET.

Para resolver o problema, criei uma classe descendente de *DataSet* que é capaz de acessar um servidor DataSnap, chamada apropriadamente de *MidasDataSet*. Eu adicionei nessa classe algumas propriedades básicas para comunicação Midas, como você pode ver no código as propriedades *RemoteServer* e *ProviderName*, como se fosse um *ClientDataSet*.

A classe se comunica como um servidor COM+ por meio da interface *IAppServer* e recebe os dados em *DataPackets*, transformando-os em um *DataTable* do ADO.NET que fica interno ao *DataSet*.

Com isso, podemos fazer o *DataBind* para um *DataGrid*. Para facilitar, eu coloquei a classe *MidasDataSet* em uma unit separada, explicada a seguir.

A classe MidasDataSet para comunicação COM+/.NET

Para implementar a classe uma nova unit deve ser criada, chamada "MidasAdoDataSet.pas". O seu código é mostrado na **Listagem 5**.

> **Nota:** Para facilitar e reduzir o código, declarei alguns membros como atributos simples na classe, mas o ideal é que você crie-os na forma de propriedades.

Para acessar o servidor DataSnap, foi necessário importar alguns namespaces da VCL for .NET, mostrados no início da unit. Você precisa referenciar os *assemblys* a partir do *Project Manager*, através da opção *Add Reference*. Os *assemblys* necessários são *Borland.VclDbRtl.dll, Borland.VclDSnap.dll e Borland.VclDSnapCon.dll*.

O método *Open* não faz muita coisa, ele na verdade chama o método interno *OpenCds* e chama o *Convert* para converter os dados de *DataPacket* para *DataSet*. O *OpenCds* instancia um *MidasConnection* (que é um *wrapper* para um *TDComConnection*) para conectar ao servidor COM+ e utiliza um *ClientDataSet* interno para armazenar os dados obtidos.

O método *Convert* faz a varredura no *DataPacket* obtido (*ClientDataSet*) e armazena em um *DataTable*, que pode ser acessado diretamente pela coleção *Tables*, pois nossa classe descende de *DataSet*. O primeiro passo é configurar as colunas e a seguir obter os registros. O *ApplyUpdates* é explicado a seguir.

Listagem 3. Método para obter os dados

```
procedure TWebForm1.btSelect_Click(
  sender: System.Object; e: System.EventArgs);
var
  ds: MidasDataSet;
begin
  ds := MidasDataSet.Create();
  ds.RemoteServer := 'ServerCOMPlus.DM';
  ds.ProviderName := 'dspCustomers';
  ds.Param := tbFInd.Text;
  ds.Open(true);
  session['ds'] := ds;
  GridDataBind();
end;
```

Listagem 4. Método para preencher o DataGrid

```
procedure TWebForm1.GridDataBind;
begin
  DG.DataSource := session['ds'] as DataSet;
  DG.DataBind();
end;
```

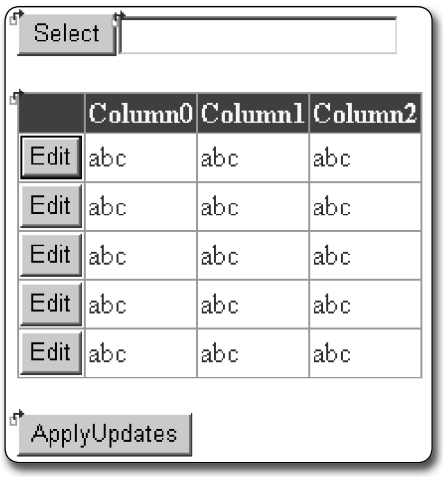


Figura 4. Web Form principal da aplicação

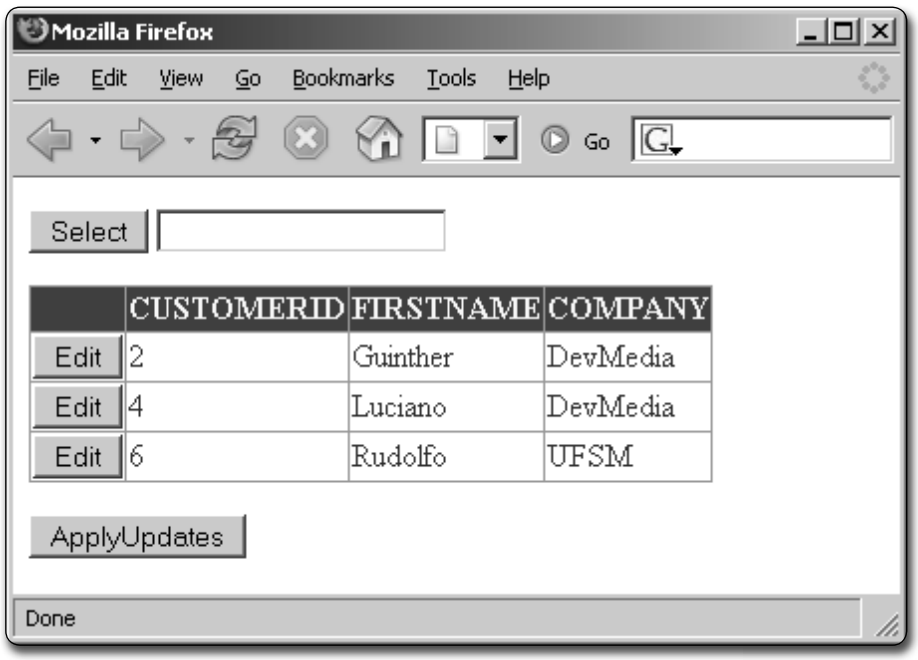


Figura 5. Aplicação ASP.NET acessando servidor DataSnap

Listagem 5. Código da unit MidasAdoDataSet

```
unit MidasAdoDataSet;
interface
uses
 Borland.Vcl.MConnect, Borland.Vcl.Db, Borland.Vcl.DBClient, Borland.Vcl.Variants, System.Data;
type
  MidasDataSet = class (DataSet)
  private Cds: TClientDataSet;
  procedure Convert();
  procedure OpenCds();
  public
    RemoteServer: string;
    ProviderName: string;
    Param: string;
    procedure Open(KeepCdsInSession: boolean);
    procedure ApplyUpdates();
end;
MidasConnection = class(System.&Object)
  public
    DCom: TDCOMConnection;
  public
    constructor Create(ServerName: string);
end;
implementation
procedure MidasDataSet.ApplyUpdates;
var
  row: DataRow;
  Key: TField;
  i: Integer;
begin
  for row in Tables[0].Rows do
    if row.RowState = DataRowState.Modified then
    begin
      Key := cds.Fields[0];
      cds.Locate(Key.FieldName, VarArrayOf([row[key.FieldName].ToString()]), []);
      cds.Edit();
      for i := 1 to cds.FieldCount - 1 do
        cds.Fields[i].AsString := row[cds.Fields[i].FieldName].ToString();
        cds.Post();
    end;
    { Testar aqui as opções de insert e delete }
  cds.ApplyUpdates(0);
end;
procedure MidasDataSet.Convert();
var
  i: integer;
  col: DataColumn;
  dt: DataTable;
  row: DataRow;
begin
  cds.Open();
  dt := DataTable.Create();
  Tables.Clear();
  Tables.Add(dt);
  for i := 0 to cds.Fields.Count - 1 do
  begin
    col := DataColumn.Create;
    col.ColumnName := cds.Fields[i].FieldName;
    dt.Columns.Add(col);
  end;
  while not cds.eof do
  begin
    row := dt.NewRow;
    for i := 0 to Cds.FieldCount - 1 do
    begin
      row[cds.Fields[i].FieldName] := cds.Fields[i].AsString;
    end;
    dt.Rows.Add(row);
    cds.Next;
  end;
  AcceptChanges();
end;
procedure MidasDataSet.Open(KeepCdsInSession: boolean);
begin
  OpenCds();
  Convert();
  if not KeepCdsInSession then cds.Free();
end;
procedure MidasDataSet.OpenCds();
var
  Con: MidasConnection;
begin
  con := MidasConnection.Create(RemoteServer);
  cds := TClientDataSet.Create(nil);
  cds.ProviderName := ProviderName;
  Cds.RemoteServer := Con.DCom;
  Cds.FetchParams();
  if Param <> '' then Cds.Params[0].AsString := Param;
  Cds.Open();
end;
constructor MidasConnection.Create(ServerName: string);
begin
  inherited Create();
  DCom := TDCOMConnection.Create(nil);
  DCom.ServerName := ServerName;
end;
end.
```

> **Nota**: Se o leitor observar, o que fiz aqui foi exatamente o inverso da técnica que demonstrei na edição passada, onde na ocasião foi preciso transformar um *DataSet* do ADO.NET em um *Data Packet*.

Executando a aplicação e clicando no botão, já podemos testar o sistema (**Figura 5)**. Você pode fornecer um parâmetro para o *like* implementado no servidor, digitando um texto no *TextBox*. Para simplificar o exemplo, a classe *MidasDataSet* considera que o *DataSet* no servidor possui nenhum ou um único parâmetro.

> **Nota**: Se a aplicação não rodou, verifique atentamente todos os passos realizados e também o código. Nesse tipo de solução, a mensagem de erro que você obtém pode não indicar o erro exato. Experimente trocar simplesmente o nome do *ProviderName* no código e veja que receberá a mirabolante mensagem "Catastrophic failure" ou "Falha Catastrófica".

Edição e ApplyUpdates

Após os dados serem obtidos do servidor, pela nossa classe *MidasDataSet*, eles são guardados em sessão (memória) para poderem ser manipulados pelo *DataGrid* do ASP.NET. Para isso, usamos o objeto *Session*.

Se observar bem, quando colocado em memória, nosso *MidasDataSet* terá dois *ResultSets*: o *ClientDataSet* originalmente obtido a partir do *DataPacket* e os dados do *DataTable*, que são as informações convertidas para ADO.NET.

Guardamos o *ClientDataSet* original em *cache* juntamente com o *MidasDataSet* por um único motivo: precisamos dele no *ApplyUpdates*. Quando chamamos o *ApplyUpdates* do nosso *MidasDataSet*, todos os valores alterados no *DataTable* são repassados de volta ao *ClientDataSet*, e então voltamos a fazer uma comunicação DataSnap com o servidor de aplicação.

> **Nota**: Caso não vá alterar o *DataSet*, usando-o apenas para consulta, passe *False* para o parâmetro *KeepCdsInSession* do *Open*. Criei esse parâmetro com o propósito de otimizar a solução caso não vá precisar editar os dados.

Para testar o *Apply*, para habilitar a edição na interface da aplicação, primeiramente crie o código da **Listagem 6** para o *EditCommand* e *CancelCommand* do *DataGrid*. O código simplesmente coloca o grid em edição baseado na célula que o usuário clicou (botão *Edit*).

> **Nota**: O *DataGrid* do ASP.NET, na sua versão 1.1, não suporta a edição automática como estamos acostumados a fazer com um *DBGrid* na VCL. Então tudo deve ser feito manualmente, como obter os dados editados e jogar no *DataSet*.

Quando o usuário clica no *Update* após a edição de um registro no Grid, precisamos capturar os valores digitados nas células e devolver para o respectivo registro no *Data Set* (nesse caso o *MidasDataSet*). Isso é feito no evento *UpdateCommand* do *DataGrid* (**Listagem 7**). O código do *Apply* simplesmente chama o *ApplyUpdates* do *MidasDataSet* que estava em sessão (**Listagem 8**). A **Figura 6** mostra a edição em execução.

> **Nota**: Para não estender o exemplo, eu omiti as operações de inserção e exclusão. Mas você pode facilmente se basear no código do *Update* para criar essas operações.

Conclusão

O exemplo aqui demonstrado é simples, mas ilustra o poder da combinação proposta. Como possíveis melhorias, sugiro que o leitor incremente o *MidasDataSet* para que se pareça ainda mais com um *ClientDataSet*, usando e abusando dos recursos de *IAppServer*. O primeiro passo está dado.

A solução aqui apresentada vai agora permitir que você crie soluções Web baseadas em ASP.NET reaproveitando todo o código da sua camada intermediária, ao mesmo tempo que mantém a compatibilidade com clientes desktop.

Dessa forma, poderá oferecer ambas as soluções, por exemplo, permitindo o acesso Win32 a partir de uma intranet, e o acesso Web a partir da Internet. ●

Listagem 6. Eventos EditCommand e CancelCommand do DataGrid

```
procedure TWebForm1.DG_EditCommand(
  source: System.Object;
  e: System.Web.UI.WebControls.DataGridCommandEventArgs);
begin
  DG.EditItemIndex := e.Item.ItemIndex;
  GridDataBind();
end;

procedure TWebForm1.DG_CancelCommand(
 source: System.Object;
 e: System.Web.UI.WebControls.DataGridCommandEventArgs);
begin
  DG.EditItemIndex := -1;
  GridDataBind();
end;
```

Listagem 7. Evento UpdateCommand do DataGrid

```
procedure TWebForm1.DG_UpdateCommand(
   source: System.Object;
   e: System.Web.UI.WebControls.DataGridCommandEventArgs);
var
  ds: DataSet;
  row: DataRow;
begin
  ds := session['ds'] as DataSet;
  row := ds.tables[0].rows[e.Item.ItemIndex];
  row['FirstName'] := (e.Item.Cells[2].Controls[0] as TextBox).Text;
  row['Company'] := (e.Item.Cells[3].Controls[0] as TextBox).Text;
  DG.EditItemIndex := -1;
  GridDataBind();
end;
```

Listagem 8. ApplyUpdates

```
procedure TWebForm1.btApply_Click(
   sender: System.Object; e: System.EventArgs);
begin
  (Session['ds'] as MidasDataSet).ApplyUpdates();
  GridDataBind();
end;
```

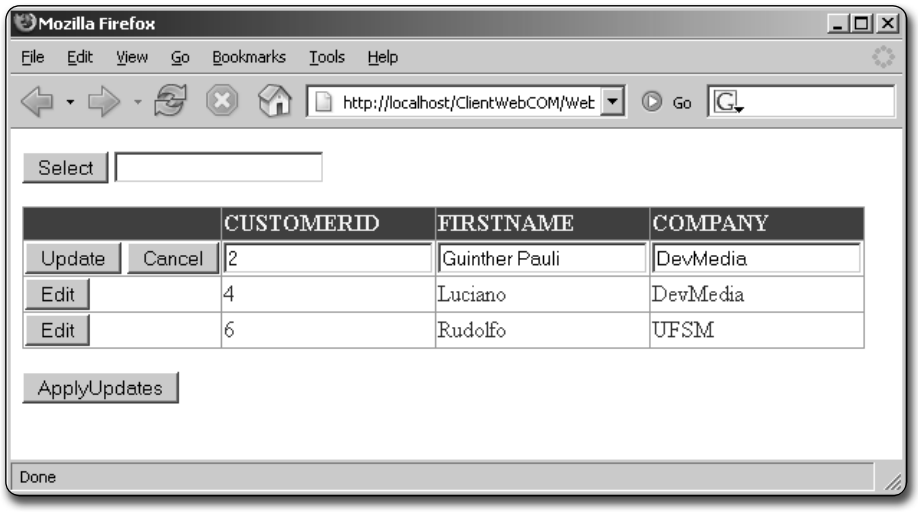


Figura 6. Testando a edição

Links

Treinamento On-Line em ASP.NET e Delphi 2005/2006 da ClubeDelphi
www.devmedia.com.br/curso/ecommerce2005

Performance no Firebird e ADO.NET

DataReaders, DataSets, Connection Pooling, Cache e Stored Procedures

Guinther Pauli

(guinther@devmedia.com.br)

é autor de mais de 100 artigos publicados e 200 vídeo-aulas e do livro "Delphi – Programação para Banco de Dados e Web". É Bacharel em Sistemas de Informação, Microsoft Certified: MCP, MCAD, MCSD.NET, Borland Certified: Delphi 6, 7, 2005, 2006, Web e Kylix. Editor Geral das Revistas .net Magazine, ClubeDelphi, WebMobile (.NET) e Mr.Bool. Palestrante em vários eventos pelo Brasil, como TechDay SP,RJ,POA,BH, Web Days e todas as edições da Borland Conference.

Neste artigo destaco algumas dicas interessantes para otimizar aplicações ASP.NET com Firebird. Veremos como a tecnologia oferece poderosos recursos para tornar suas aplicações Web robustas e escaláveis, usando o mínimo de esforço possível. Conheceremos os poderosos recursos de cache de dados, uso efetivo de *Stored Procedures*, *Connection Pooling* e outras técnicas avançadas.

Você aprenderá como usar *DataSets* em memória para evitar consultas desnecessárias ao servidor SQL e otimizar assim o tráfego de dados. Você também conhecerá um pouco sobre o interessante recurso de *Connection Pooling* do ADO.NET. Para construir os exemplos, utilizarei o Delphi 2006 e o Firebird 2.0 como banco de dados.

Connection Pooling

Inicie uma nova aplicação ASP.NET no Delphi 2006. A partir da *Component Palette*, coloque um *FbConnection* no Web Form. Selecione o componente e no *Object Inspector* acesse o editor da propriedade

Provider ADO.NET Para Firebird

Para acessar o Firebird no ASP.NET, você pode utilizar o provider ADO.NET do próprio Firebird. Para baixá-lo, utilize o seguinte endereço: www.firebirdsql.org/index.php?op=files&id=netprovider. A versão utilizada é para o .NET Framework 1.1. A instalação é bastante simples, basta executar o instalador.

Para instalar no Delphi 2006, acesse o menu Component>Installed Components. No editor, digite "Firebird" em Category, clique no botão Select an Assembly e escolha o arquivo FirebirdSql.Data.Firebird.dll, que por padrão encontra-se em: C:\Arquivos de programas\FirebirdNETProvider1.7. Clique em OK e veja na Tool Palette os componentes instalados para acesso ao Firebird.

ConnectionString. No editor que aparece informe os parâmetros para acesso ao banco *Employee.fdb* do Firebird.

Com isso, configuramos a conexão ao Firebird usando o provider nativo, a primeira dica de performance (jamais use OleDB, ODBC ou outro provider nesse caso).

Observe que em *User Name* e *Password* informamos um usuário e senha padrão para acesso ao banco. Aqui vai a segunda dica valiosa para otimização: forneça um usuário e senha fixos, de forma que todos os usuários que conectem à aplicação utilizem as mesmas credenciais.

Se for necessário restringir acesso a determinado usuário, defina isso na forma de autorizações no *Web.config*. Fornecer um usuário fixo fará o ADO.NET utilizar de forma efetiva o recurso de *Connection Pooling*, sem ter perda de desempenho.

Connection Pooling é o mecanismo que permite ao ADO.NET reaproveitar conexões ao banco de dados. Imagine a seguinte situação: um usuário acessa a aplicação, conectamos ao BD para extrair informações e a exibimos no formulário. A seguir, fechamos a conexão e devolvemos o resultado ao browser.

Como aplicações Web são *state less* (sem estado), se esse mesmo ou outro usuário se conectar à aplicação, uma nova conexão precisará ser restabelecida. Conectar ao BD a cada requisição de usuário é literalmente um suicídio em ambiente Web, onde uma aplicação pode ter centenas e até milhares de conexões simultâneas.

O ADO.NET resolve isso de forma bastante elegante: após a página ser enviada ao browser, a conexão com o BD não é liberada, mesmo que você tenha chamado explicitamente o método *Close* do *Fb Connection*. O ADO.NET guarda automaticamente a conexão em *pool* (imagine isso como uma espécie de *cache* de conexões).

Ou seja, a conexão fica aberta com o banco de dados e persiste entre requisições. Quando outro usuário conectar na aplicação, o ADO.NET verifica se existe uma conexão disponível no *pool* e caso encontre, a utiliza. Com isso, todo o tempo necessário para localizar o servidor de BD, estabelecer uma conexão, autenticar um usuário e verificar permissões não será mais consumido a cada requisição.

E o melhor de tudo, você não precisa fazer nada para usar esse recurso, pois ele já é ativado por padrão. Criar um mecanismo de *Connection Pooling* "no braço" via código é algo extremamente complicado (infelizmente já tive que passar por esse esforço em uma ocasião). No ADO.NET já temos isso pronto no próprio framework. Produtividade é um dos pontos fortes do .NET.

> **Nota:** O *Connection Pooling* só pode ser usado em ambiente *multi-thread* (uma aplicação Web, por exemplo), onde temos várias *threads* simultâneas processando solicitações clientes. Não faz sentido, por exemplo, usar *Connection Pooling* em uma aplicação Windows Forms tradicional (duas camadas). Internamente, o *Connection Pooling* faz uso de um interessante recurso do sistema operacional Windows para prover sua funcionalidade: *semáforos*.

Você pode ainda controlar como o ADO.NET trabalha com *Connection Pooling*, fazendo alguns ajustes na propriedade *ConnectionString* do *FbConnection* ou diretamente no editor de conexões da **Figura 1**. Podemos especificar alguns parâmetros, como mostra a **Tabela 1**.

Veja um exemplo de como poderíamos usar alguns desses parâmetros na *string* de conexão da nossa aplicação (você pode fazer essa modificação diretamente na propriedade *ConnectionString*, a partir do *Object Inspector*) com o código da **Listagem 1**.

> **Atenção**: Sempre use a mesma *Connection String* (com os mesmos valores para todos os parâmetros) em todos os objetos de conexão, para que compartilhem o mesmo *Connection Pooling*.

Listagem 1. Parâmetros na string de conexão

```
User=SYSDBA;Password=masterkey;Database=C:\Program Files\Firebird\Firebird_2_0\examples\
     empbuild\EMPLOYEE.FDB;
DataSource=LOCALHOST;Port=3050;Dialect=3;Charset=NONE;
Role=;Packet Size=8192;Server Type=0;Pooling=True;
Min Pool Size=50;Connection Lifetime=120;
Connection timeout=15;
```

Parâmetro	Valor Default	Descrição
Connection Lifetime	0	Tempo de vida, em segundos, que uma conexão deve ficar no *pool* desde a sua criação.
Max Pool Size	100	Número máximo de conexões que podem ficar em *pool*.
Min Pool Size	0	Número mínimo de conexões que devem ficar em *pool*.
Pooling	True	Indica se o *Connection Pooling* está habilitado.

Tabela 1. Parâmetros do Connection Pooling

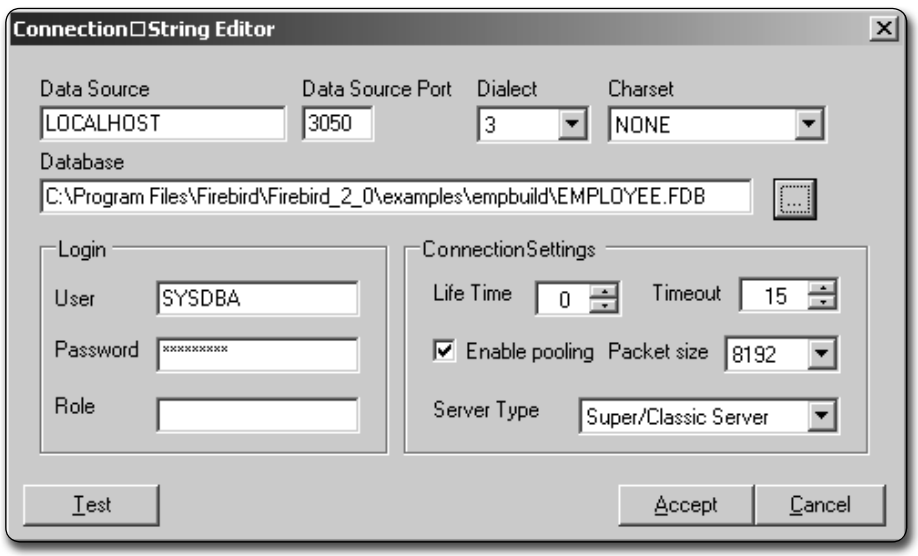


Figura 1. Parâmetros de conexão ao Firebird

Usando um DataReader para exibir dados em um DataGrid

Seguindo nosso exemplo, veremos em um primeiro momento como exibir dados de uma tabela do Firebird em um *DataGrid* (coloque um no form), usando nossa conexão previamente configurada.

A maneira mais rápida de extrair dados do banco é usando um *DataReader* (um *FbDataReader*, no caso do provider para Firebird). Um *DataReader* é responsável por fazer a leitura dos dados retornados por uma consulta SQL, usando um cursor de dados. Ele é rápido pelos seguintes motivos:

- É unidirecional: a navegação pelos registros é feita de forma seqüencial (*forward-only*). Basicamente lemos um registro, fazemos alguma coisa com ele (exibimos dados em um controle, por exemplo) e navegamos para o próximo registro;
- É somente-leitura: não é possível alterar registros de um *DataReader*;
- Não faz cache: após um registro ser lido, ele é descartado da memória.

O código da **Listagem 2** mostra como usar um *DataReader* (coloque o código no *Page_Load* do Web Form). Aqui abrimos a conexão com o banco usando o método *Open* do *FbConnection*. A seguir, usamos um *FbCommand* para executar um comando *Select* na tabela *Employee*.

Para executar a consulta, chamamos o *ExecuteReader* do *FbCommand* (chamado aqui de *cmd*), cujo valor de retorno é atribuído à propriedade *DataSource* do *DataGrid*.

Mas onde está o *FbDataReader?* Um *DataReader* nunca é instanciado diretamente. Você sempre obterá a referência a um objeto desse tipo através da chamada ao *ExecuteReader* de um *FbCommand*. Você pode então usar o objeto de retorno para varrer o *resultset* obtido, usando o *Read*.

Aqui não foi necessário fazer essa varredura manualmente, visto que usamos o recurso de *DataBind* do ASP.NET para vinculação de dados com o *DataGrid*, através da propriedade *DataSource*.

Observe que todo o código de execução é envolvido em um bloco *try...finally*. O código dentro do bloco *finally* sempre será executado, incondicionalmente, mesmo que uma exceção ocorra (por exemplo, se o comando SQL digitado fosse inválido).

Com isso garantimos que o *Close* do *FbConnection* sempre seja chamado após a execução do código dentro do *try*, devolvendo a conexão para o *pool*. Uma outra dica é sempre manter o mínimo de código possível entre o *Open* e o *Close*, somente o que fizer uso da conexão aberta.

Usando DataSets em Cache

Sem dúvida o principal componente do ADO.NET é o *DataSet*. Ele pode ser usado com qualquer um dos providers para ADO.NET, representando uma estrutura de dados em memória. Aqui o termo "memória" se refere ao fato de que podemos criar uma consulta a uma tabela do banco de dados, extrair informações e usá-las após a conexão ser fechada. Por esse motivo, aplicações que usam esse componente são conhecidas como "desconectadas".

Enquanto um *DataReader* exige que uma conexão esteja ativa para que uma leitura de dados seja feita, um *DataSet* pode obter seus dados uma única vez e guardá-los em memória para uso posterior. Isso garante ainda mais a escalabilidade de aplicações ASP.NET. Depende de você detectar quando é melhor usar um *DataReader* ou um *DataSet*.

Para ficar mais claro, vamos imaginar uma situação: você trabalha para uma universidade e precisa construir a página de inscrição para o concurso vestibular promovido pela instituição. Nessa página, existe um formulário para cadastro do candidato, onde constam alguns *TextBoxes* para entrada de campos como *Nome*, *Endereço*, *Data Nasc* etc.

A única informação dinâmica (que vem de um BD) está em um *DropDownList*, onde o candidato pode escolher o curso que deseja se inscrever. Essas informações são obtidas a partir de uma tabela do banco de dados, que possui todos os cursos com vagas disponíveis.

Agora responda uma coisa: para que conectar ao banco de dados para preencher o *DropDownList*, a cada vez que um usuário abre a página, se os cursos nunca mudam? Em uma situação como essa, a tabela de cursos seria modificada no BD provavelmente uma vez por ano ou semestre. Esse é um exemplo típico onde podemos usar um *DataSet* ao invés de um *DataReader*.

Vamos ver como usar o recurso na prática. Inicie uma nova aplicação ASP.NET, seguindo os mesmos passos do exemplo anterior. Coloque um *FbDataAdapter* no formulário e no assistente que abrirá configure uma conexão ao banco Firebird como fizemos anteriormente. Utilize a seguinte instrução SQL para obter dados da tabela *Departments*:

```
select * from DEPARTMENT
```

Coloque um *DropDownList* no formulário e no *Page_Load* do Web Form digite o código apresentado na **Listagem 3**.

Aqui estamos simplesmente testando se a página está sendo carregada pela primeira vez (*IsPostBack*) para então inicializar o *DropDownList*. Observe

Listagem 2. Utilizando o DataReader

```
procedure TWebForm1.Page_Load(sender: System.Object; e: System.EventArgs);
var
  cmd: FbCommand;
begin
  FbConnection1.Open();
  try
    cmd := FbCommand.Create('select * from Employee', FbConnection1);
    DataGrid1.DataSource := cmd.ExecuteReader();
    DataGrid1.DataBind();
  finally
    FbConnection1.Close();
  end;
end;
```

Listagem 3. Código do Page_Load

```
procedure TWebForm1.Page_Load(sender: System.Object; e: System.EventArgs);
begin
  if (not IsPostBack) then
  begin
    DropDownList1.DataSource := GetDepartments();
    DropDownList1.DataTextField := 'Department';
    DropDownList1.DataValueField := 'DepartmentId';
    DropDownList1.DataBind();
  end;
end;
```

que atribuímos o valor da propriedade *DataSource* para uma função chamada *GetDepartments* (vista na **Listagem 4**), que retorna um *DataSet*. Implemente a função *GetDepartments* conforme mostrado na **Listagem 4**.

No código anterior, testamos se existe na *Cache* uma variável com o nome que estipulamos ("Department"). Se existir, é porque o *DataSet* já está em memória (provavelmente um usuário já realizou a consulta anteriormente). Caso contrário, refazemos a *cache* de dados, chamando o *Fill* do *FbDataAdapter* para preencher os dados do *DataSet*. E finalmente, colocamos o *DataSet* em memória (objeto *Cache*).

O *Cache* é usado no ASP.NET para compartilhar dados entre todos os usuários da aplicação. Mesmo após a requisição ser processada, as informações permanecem na memória do servidor e podem ser utilizadas posteriormente. Você pode colocar quantos objetos quiser em memória, bastando fornecer um nome diferente para cada um, mas tome cuidado para não exagerar na quantidade de informações que você vai armazenar no servidor.

> **Nota:** O ASP.NET permite que você use um *state server* (servidor de estado), permitindo que as informações colocadas em *cache* e sessão residam em um processo diferente do *aspnet_wp.exe* (o processo usado pelo ASP.NET para rodar as aplicações Web, no WinXP). É possível, inclusive, especificar um servidor dedicado (que tenha mais memória, por exemplo) apenas para armazenar objetos de cache e sessão. Uma outra alternativa, é persistir as sessões no próprio banco de dados, para poupar memória. *State Servers* também são usados para compartilhar dados de sessão quando múltiplos servidores são usados.

A **Figura 2** mostra a aplicação em execução. Faça um teste, abra um segundo browser e acesse a mesma página, porém antes pare o servidor de banco de dados. Observe que os dados serão mostrados mesmo que o Firebird esteja desativado, indicando que as informações realmente não foram obtidas a partir do BD, mas a partir da *cache* que já estava armazenada na memória do servidor.

Listagem 4. Função GetDepartments

```
public
  function GetDepartments(): DataSet;
...
function TWebForm1.GetDepartments(): DataSet;
var
  ds: DataSet;
begin
  if (Cache['Department'] = nil) then
  begin
    ds := DataSet.Create();
    FbDataAdapter1.Fill(ds,'Department');
    Cache['Department'] := ds;
  end;
  Result := Cache['Department'] as DataSet;
end;
```

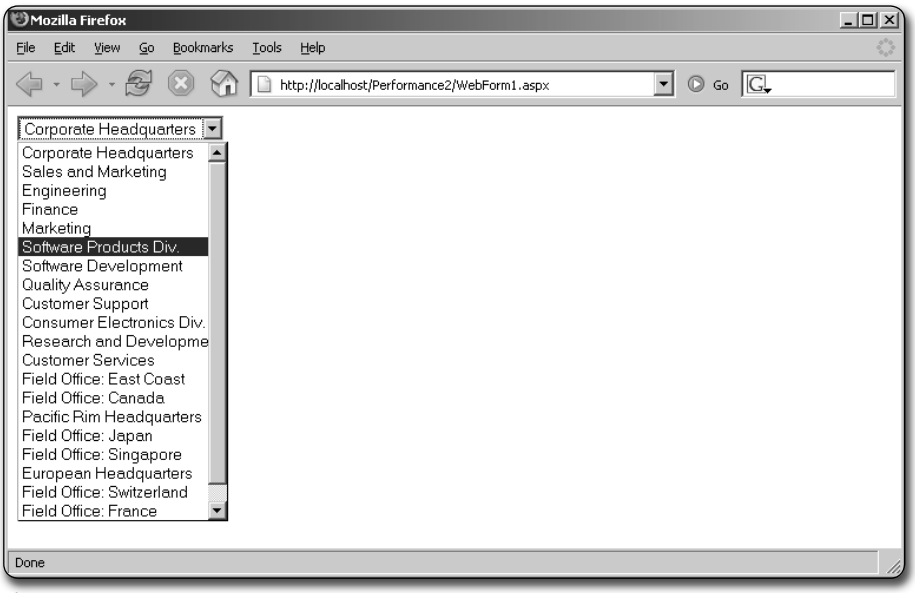


Figura 2. Usando um DataSet em Cache

Usando DataViews

Vamos aprimorar o exemplo. Vamos permitir que o usuário veja os empregados relacionados quando escolher um determinado departamento. Ao invés de criarmos uma nova consulta ao BD após a seleção, vamos filtrar uma consulta que já estará residente em memória. Ou seja, vamos guardar todos os empregados em *cache* e usar um *DataView* para filtragem.

Coloque um *ListBox* no formulário e um segundo *FbDataAdapter*, configurando a seguinte instrução SQL, que retorna todos os registros da tabela de *Employee*:

```
select * from EMPLOYEE
```

Ao invés de utilizar um segundo *DataSet* para guardar os produtos, vamos usar o mesmo criado anteriormente. Isso é possível de ser feito no ADO.NET, basta passar o mesmo *DataSet* como parâmetro para o *Fill* de ambos os *FbDataAdapters*.

Inclua então o seguinte código, logo após o *Fill* do *FbDataAdapter1*, que está no *GetDepartments*:

```
...
FbDataAdapter2.Fill(ds, 'Employee');
```

Agora, quando a página for aberta pela primeira vez, o *DataSet* conterá em memória dados sobre todos os empregados e departamentos do banco de dados. Tudo o que precisamos fazer é manipular essas informações da melhor forma, sem a necessidade de consultar o BD a cada requisição.

Por exemplo, para exibir os empregados relacionados ao departamento selecionado, altere o *Page_Load* como mostrado na **Listagem 5**.

O código é semelhante ao usado para preencher o *DropDownList*, exceto que agora estamos usando um *DataView* como fonte de dados para o controle *ListBox*. A função *GetEmployees* recebe o código de um departamento por parâmetro e

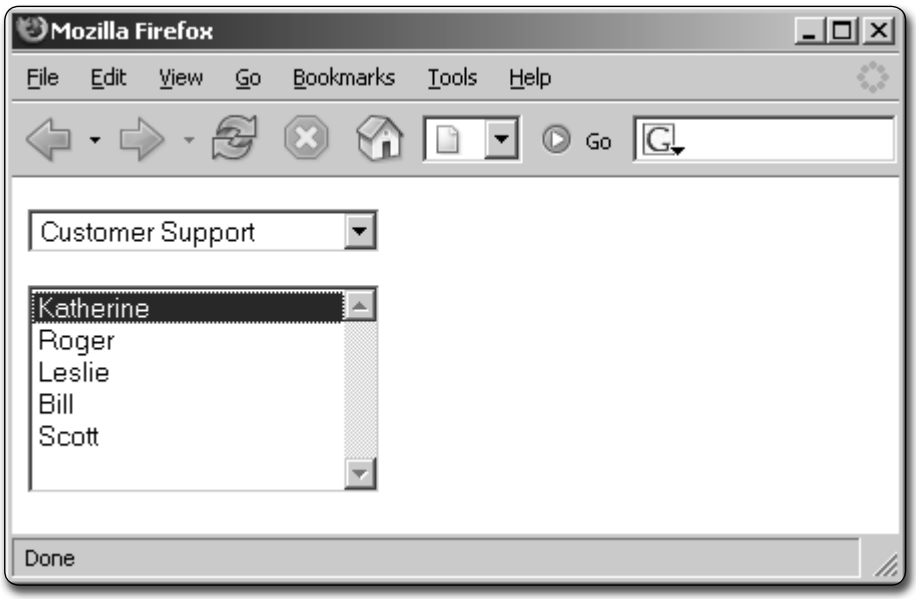


Figura 3. Usando DataViews a partir de um DataSet em memória

Listagem 5. Código do Page_Load, atualizado

```
procedure TWebForm1.Page_Load(sender: System.Object; e: System.EventArgs);
begin
  if (not IsPostBack) then
  begin
    { código do DropDownList }
  end
  else
  begin
    ListBox1.DataSource := GetEmployees(DropDownList1.SelectedValue);
    ListBox1.DataTextField := 'First_Name';
    ListBox1.DataBind();
  end;
end;
```

Listagem 6. Função GetEmployees

```
public
  function GetEmployees(Dept_No: System.&Object): DataView;
...
function TWebForm1.GetEmployees(Dept_No: TObject): DataView;
var
  dv: DataView;
begin
  dv := DataView.Create();
  dv.Table := (Cache['Department'] as
    DataSet).Tables['Employee'] as DataTable;
  dv.RowFilter := 'Dept_No = ' + Dept_No.ToString();
  Result := dv;
end;
```

Listagem 7. Stored Procedure para inserção

```
CREATE PROCEDURE COUNTRY_I (
  COUNTRY VARCHAR(15),
  CURRENCY VARCHAR(10))
AS
BEGIN
  INSERT INTO COUNTRY (COUNTRY, CURRENCY)
  VALUES (:COUNTRY, :CURRENCY);
END
```

Listagem 8. Executando a Stored Procedure

```
procedure TWebForm1.Button1_Click(sender: TObject; e: System.EventArgs);
begin
  FbConnection1.Open();
  try
    FbCommand1.Parameters[0].Value := TextBox1.Text;
    FbCommand1.Parameters[1].Value := TextBox2.Text;
    FbCommand1.ExecuteNonQuery();
  finally
    FbConnection1.Close();
  end;
end;
```

retorna os empregados relacionados, através de um *DataView*, obtidos a partir do *DataSet* de memória. Seu código é mostrado na **Listagem 6**.

Um *DataView* é ideal para se usar com *DataSets* de memória. Ele permite que você filtre e ordene *resultsets*, fornecendo diferentes visões de um mesmo conjunto de dados, sem usar nenhum tipo de instrução SQL ou comunicação extra com o banco. Vale lembrar que você pode ter vários *DataViews* atuando sobre um mesmo *DataSet*.

O último passo é configurar o *AutoPostBack* do *DropDownList* para *True*, para que seja feito o *postback* no servidor quando o usuário escolher um item no controle. A **Figura 3** mostra o exemplo em execução.

É claro, se os dados forem modificados no banco de dados, eles não serão refletidos no *DataSet* da cache. É sua obrigação refazer a cache quando necessário. O ASP.NET possui alguns recursos prontos para facilitar esse processo. Podemos, por exemplo, especificar um critério de expiração para a cache ou criar um mecanismo de dependência.

Inserindo com Stored Procedures

Não poderíamos falar em otimizações de aplicações ASP.NET com ADO.NET sem mostrar a utilização de *Stored Procedures*. Elas aumentam drasticamente a velocidade de soluções Web, tanto na obtenção de dados do Firebird quanto na atualização, inserção ou exclusão de registros.

Isso acontece porque o servidor SQL pode otimizar planos de execução e pré-compilar essas instruções que residem no BD, e não mais na aplicação cliente, que apenas se encarrega de passar os parâmetros apropriados.

Neste exemplo, construiremos uma aplicação em ASP.NET que fará uso efetivo de *Stored Procedures*, de forma a comprovar o aumento de escalabilidade. Primeiramente, no banco de dados, crie a *Stored Procedure* da **Listagem 7**, para inserir dados na tabela *Country*.

Em uma nova aplicação ASP.NET, vamos colocar no Web Form alguns *TextBoxes* para a entrada de dados. Para facilitar, vamos utilizar uma tabela simples do banco *Employee*, a tabela *Country*,

além de trabalhar apenas com a operação de inserção. Utilizando *Labels*, *Buttons* e *TextBox*, desenhe um formulário igual ao da **Figura 4**.

A seguir, coloque um *FbConnection* e configure uma conexão ao FB como feito anteriormente. Coloque um *FbCommand* e aponte a propriedade *Connection* para o *FbConnection*. Defina seu *CommandType* para *StoredProcedure* e *CommandText* referencie a SP criada ("COUNTRY_I").

Em *Parameters* inclua dois parâmetros, como mostra a **Figura 5**. Observando o que foi feito para o parâmetro *Country*, faça o mesmo para o parâmetro *Currency*.

No botão *Inserir* digite o código da **Listagem 8**, para inserir os dados digitados nos *TextBoxes* no banco de dados, usando a *Stored Procedure* criada. Note que configuramos os parâmetros criados na coleção *Parameters* do *FbCommand*. *ExecuteNonQuery* faz a execução da SP propriamente dita.

Conclusão

Utilizando as técnicas vistas neste artigo, você poderá agora otimizar suas aplicações ASP.NET para que tenham uma melhor performance e escalabilidade, deixando seu usuário final muito mais satisfeito com o tempo de resposta da sua aplicação.

A utilização de *Stored Procedures*, quando feita de forma apropriada, pode garantir o sucesso de sua solução ASP.NET com ADO.NET, sendo imbatível se comparada a qualquer outra tecnologia existente. *DataSets* em memória e *DataReaders* também são ótimos ingredientes para turbinar seu Web Site com ASP.NET, ADO.NET e Firebird. ●

ClubeDelphi PLUS www.devmedia.com.br/clubedelphi/portal.asp

Acesse agora o mesmo o portal do assinante ClubeDelphi e assista a uma série de vídeo aulas de Guinther Pauli, sobre performances no ASP.NET com ADO.NET.

www.devmedia.com.br/articles/listcomp.asp?txtsearch=Performance+no+ASP.NET+com+ADO.NET

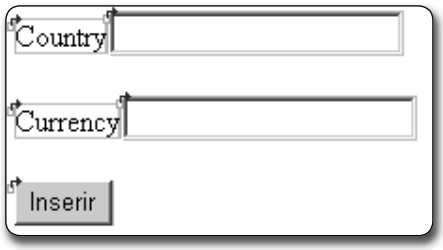
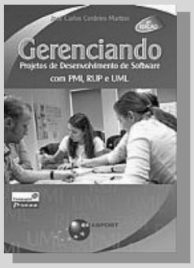


Figura 4. Formulário principal da aplicação, com TextBoxes

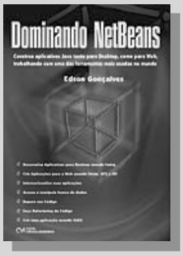
Figura 5. Inserindo parâmetros

Mais Vendidos!



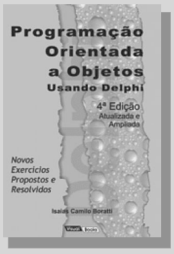
Gerenciando Projetos de Desenvolvimento de Software com PMI, RUP e UML 4ª. Edição
Brasport
R$ 75,00

Dominando Netbeans
Construa aplicativos Java tanto em Desktop, como para Web
Ciência Moderna
De:R$ 76,00
Por: R$ 72,20

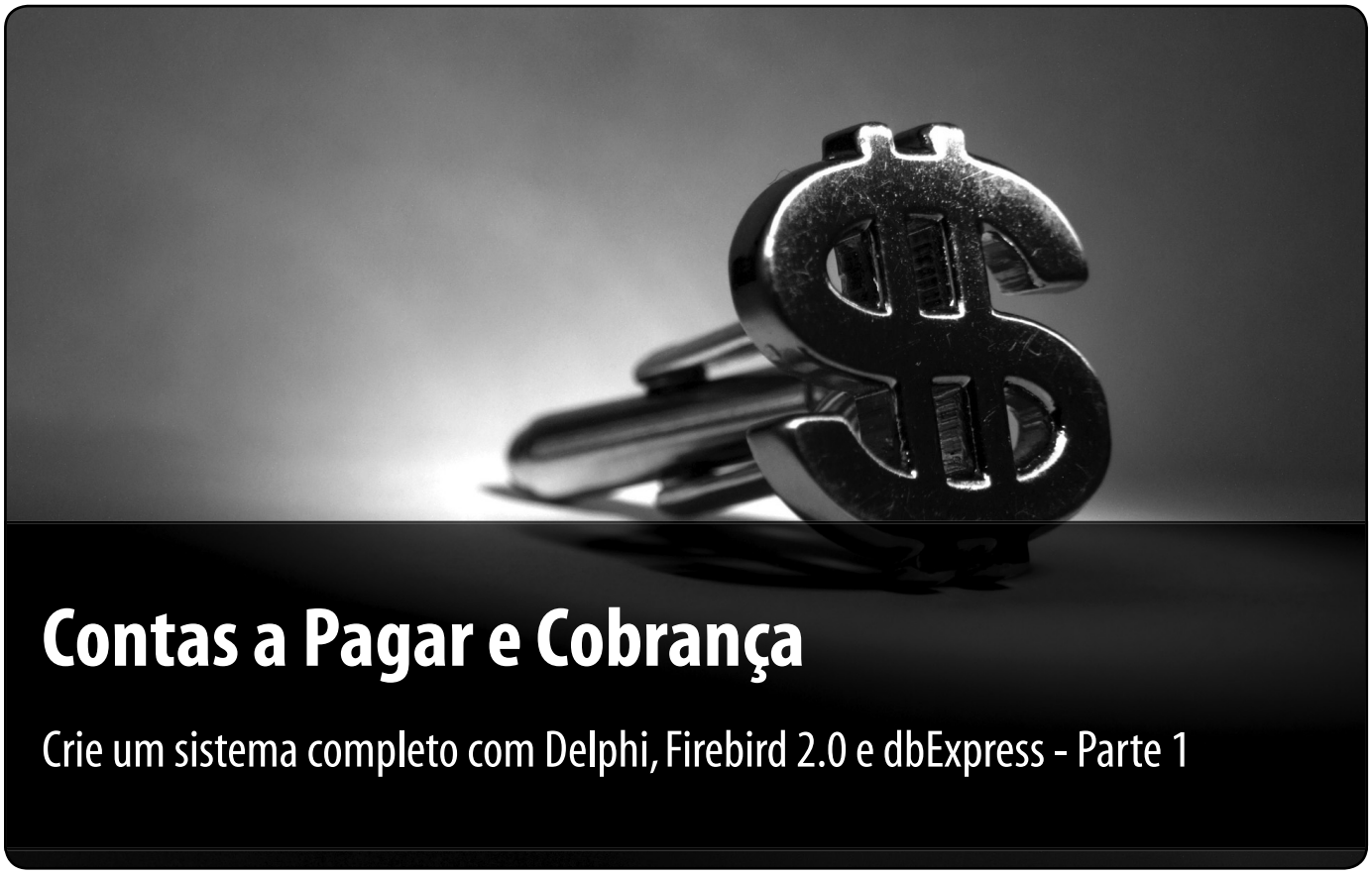


Desenvolvendo Aplicações Web com NetBeans IDE 5.5
Ciência Moderna
R$ 99,00

Programação Orientada a Objetos Usando Delphi - 4ª edição
Visual Books
De: R$ 75,00
Por: R$ 71,25



Frete Grátis Para todo o Brasil!
Até 6x sem juros no cartão.*
*Parcela mínima de R$ 20,00
Outras facilidades de pagamento:
Tel: (11) 4062-5152
livrosdeprogramacao.com.br



Contas a Pagar e Cobrança

Crie um sistema completo com Delphi, Firebird 2.0 e dbExpress - Parte 1

Adriano Santos
(artes@doiscliques.com)
é desenvolvedor Delphi desde 1998. Professor e programador PHP. Bacharel em Comunicação Social pela Universidade Cruzeiro do Sul, SP. É colunista e membro da Comissão Editorial da revista ClubeDelphi.

Bem-vindos ao novo mini-curso da ClubeDelphi, onde o leitor aprenderá como criar um sistema completo de gerenciamento de contas a pagar e receber, com emissão de boletos bancários, baixa de pagamentos, fluxo de caixa, impressão de cheques e uma série de outros recursos encontrados nesse tipo de sistema.

O sistema que criaremos terá como principal objetivo proporcionar ao usuário o controle total de contas a pagar e a receber, ou seja, será possível adicionar contas de luz, telefone, água, gastos com office-boy, compra de equipamentos etc.

Da mesma forma, o usuário poderá incluir as faturas que tem a receber de seus clientes, processo esse que chamaremos de Cobrança. Com essas informações no sistema, sempre atualizadas, poderemos emitir relatórios, gráficos, boletos bancários e fluxo de caixa para o acompanhamento das contas correntes.

Neste artigo, criaremos os principais cadastros que farão parte do exemplo, como: Clientes, Fornecedores e Contas Correntes.

Criando o banco de dados

Para iniciar o projeto, vamos primeiramente criar o banco de dados e as tabelas necessárias para cada tela de cadastro. As tabelas são: *Clientes*, *Fornecedores* e *Contas*. Neste artigo, criaremos o banco de dados utilizando a ferramenta IBExpert em sua versão *Standard*. Por isso, acesse o link *www.ibexpert.com* e em seguida entre no item *IBExpert*.

À esquerda do site do fabricante clique em *Download>Free*. Preencha o formulário de cadastro e aguarde o e-mail com as

Metodologia

O exemplo deste artigo utiliza a arquitetura cliente/servidor. Para mais informações sobre essa arquitetura e outras, visite: http://www.devmedia.com.br/articles/viewcomp.asp?comp=5219

instruções de download da ferramenta. Após isso, abra o IBExpert e vamos criar o banco de dados usando a opção *Script Executive* presente no menu *Tools*.

Com a tela de *scripts* aberta, digite o código da **Listagem 1**. Nele estão contidos os esquemas para a criação do banco de dados assim como de cada tabela do sistema. Digitado o *script*, basta executá-lo usando o botão *Run Script* ou pressione a tecla F9.

> **Nota:** Substitua *<Caminho>\SYSPAGUE.FDB* pelo diretório do banco de dados e seu nome, ex: *C:\BancoDeDados\SysPague.fdb*.

Para este exemplo criamos somente as tabelas necessárias para os cadastros que usaremos neste artigo. Note que criamos em cada tabela os campos DT_ALTERACAO e STATUS. Esses receberão a data de alteração/criação e o estado do registro respectivamente.

Esse último terá uma função especial. Não faremos exclusões no banco de dados. Cada vez que o usuário solicitar a exclusão de um registro, o mesmo terá seu STATUS alterado para "I" de inativo e não mais aparecerá em telas de consultas, cadastros ou relatórios. Dessa forma mantemos uma espécie de histórico do registro.

Formulários da aplicação

As telas de cadastro serão herdadas de um formulário padronizado que será criado. Assim, poupamos tempo de design e programação, já que boa parte do código e componentes comuns ficarão no formulário padrão.

No formulário padrão haverá uma caixa para pesquisa no banco de dados e botões comuns para Inclusão, Alteração, Exclusão etc. Além dos formulários de cadastro, uma tela principal fará a ligação entre as janelas de nosso sistema, assim poderemos acessar todos os cadastros, relatórios e ferramentas que incluiremos.

ClubeDelphi PLUS www.devmedia.com.br/clubedelphi/portal.asp

Acesse agora o mesmo o portal do assinante ClubeDelphi e assista a uma vídeo aula de Luciano Pimenta que mostra como baixar e instalar o IBExpert em sua versão gratuita.

www.devmedia.com.br/articles/viewcomp.asp?comp=3082

Após a criação do banco, podemos iniciar o desenvolvimento do sistema, onde usaremos Delphi 7.0, Firebird 2.0 e dbExpress. Abra o Delphi e crie um novo projeto através do menu *File>New>Application*.

Mude a propriedade *Name* do formulário para "frmPrincipal" e salve a unit como "uPrincipal.pas". Mude também a propriedade *BorderStyle* para *bsSingle* e desative a opção *biMaximize* em *BorderIcons*.

Por fim, modifique o *Caption* para "Sistema de Contas a Pagar e Cobrança" e o *Position* para *poScreenCenter*. Inclua agora um *ImageList* ("imgBotoes"), um *MainMenu* ("mnuPrincipal") e um *ToolBar* ("tobBotoes").

No *imgBotoes* adicione quatro imagens a sua escolha representando os botões *Clientes*, *Fornecedores*, *Contas* e *Sair*. O *mnuPrincipal* receberá três itens correspondentes aos cadastros que criaremos (**Figura 1**).

Faremos nesse instante a configuração da barra de ferramentas. Adicione quatro botões na *Toolbar* clicando com o botão

Listagem 1. Criação do banco, tabelas e índices

```
SET SQL DIALECT 3;
SET NAMES WIN1252;

CREATE DATABASE '<Caminho>\SYSPAGUE.FDB'
USER 'SYSDBA' PASSWORD 'masterkey'
PAGE_SIZE 8192
DEFAULT CHARACTER SET WIN1252;

CREATE TABLE CLIENTES (
  CNPJ          VARCHAR(18) NOT NULL,
  FANTASIA      VARCHAR(20) NOT NULL,
  RAZAO         VARCHAR(150),
  IE            VARCHAR(18) NOT NULL,
  ENDERECO      VARCHAR(100),
  BAIRRO        VARCHAR(50),
  COMPLEMENTO   VARCHAR(30),
  CIDADE        VARCHAR(50),
  ESTADO        VARCHAR(2),
  CEP           VARCHAR(9),
  TELEFONE      VARCHAR(15),
  FAX           VARCHAR(15),
  DT_CADASTRO   TIMESTAMP,
  STATUS        VARCHAR(1),
  DT_ALTERACAO  TIMESTAMP);

CREATE TABLE CONTAS (
  BANCO         INTEGER NOT NULL,
  AGENCIA       VARCHAR(10) NOT NULL,
  CONTA         VARCHAR(10) NOT NULL,
  NOME_AGENCIA  VARCHAR(30),
  NOME_CONTA    VARCHAR(30),
  ENDERECO      VARCHAR(100),
  BAIRRO        VARCHAR(50),
  COMPLEMENTO   VARCHAR(30),
  CIDADE        VARCHAR(50),
  ESTADO        VARCHAR(2),
  CEP           VARCHAR(9),
  TELEFONE      VARCHAR(15),
  FAX           VARCHAR(15),
  DT_CADASTRO   TIMESTAMP,
  CONTATO       VARCHAR(50),
  STATUS        VARCHAR(1),
  DT_ALTERACAO  TIMESTAMP);

CREATE TABLE FORNECEDORES (
  CNPJ          VARCHAR(18) NOT NULL,
  FANTASIA      VARCHAR(20) NOT NULL,
  RAZAO         VARCHAR(150),
  IE            VARCHAR(18) NOT NULL,
  ENDERECO      VARCHAR(100),
  BAIRRO        VARCHAR(50),
  COMPLEMENTO   VARCHAR(30),
  CIDADE        VARCHAR(50),
  ESTADO        VARCHAR(2),
  CEP           VARCHAR(9),
  TELEFONE      VARCHAR(15),
  FAX           VARCHAR(15),
  DT_CADASTRO   TIMESTAMP,
  STATUS        VARCHAR(1),
  DT_ALTERACAO  TIMESTAMP);

ALTER TABLE CLIENTES ADD CONSTRAINT PK_CLIENTES
PRIMARY KEY (CNPJ);

ALTER TABLE CONTAS ADD CONSTRAINT PK_CONTAS
PRIMARY KEY (BANCO, AGENCIA, CONTA);

ALTER TABLE FORNECEDORES
ADD CONSTRAINT PK_FORNECEDORES PRIMARY KEY (CNPJ);
```

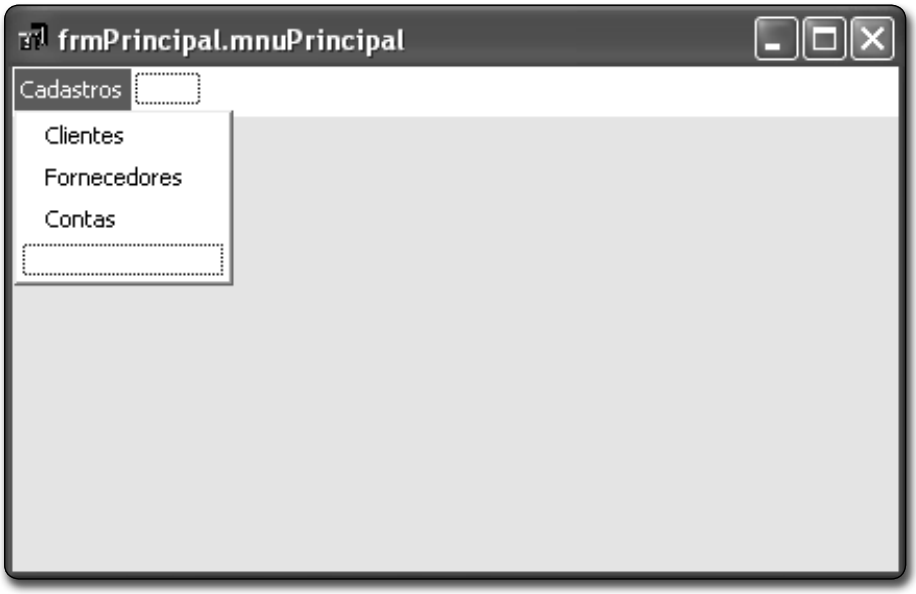


Figura 1. MainMenu

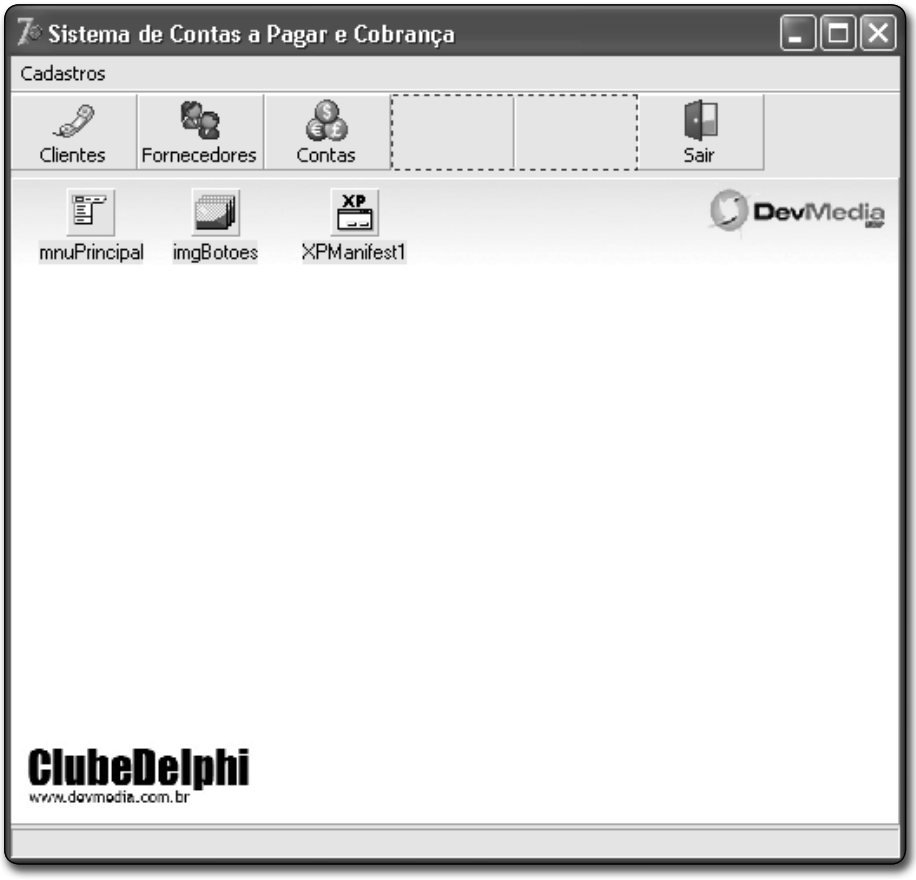


Figura 2. Tela principal da aplicação

direito e em seguida em *New Button*. Modifique a propriedade *Images* da *Toolbar* para o *imgBotoes* e então altere o *Image Index* de cada botão informando o índice da imagem presente no *ImageList*.

> **Nota:** Não esqueça de incluir o *Caption* de cada botão e ativar a propriedade *ShowCaptions* da *Toolbar*.

Daremos um toque mais profissional ao projeto incluindo um *Panel* alinhado como *alClient* no formulário principal e dentro dele (também *alClient*) um *Image*. Nesse *Image* adicione uma figura usando a propriedade *Picture*.

Se necessário modifique o *AutoSize* e/ou *Stretch* esticando ou encolhendo a imagem escolhida. Veja na **Figura 2** um exemplo de tela que desenvolvi. Salve o projeto com o nome de "SysPague.dpr".

Desenvolvendo o Data Module

Como desenvolveremos diversas telas de acesso a dados, precisamos de um Data Module que contenha as conexões necessárias para tabelas e *Stored Procedures*. A princípio não foram criadas *Stored Procedures*, porém ao longo do mini-curso criaremos SPís para manipulação ou seleção de dados para relatórios ou boletos bancários.

Vamos então iniciar o desenvolvimento do Data Module. Clique em *File>New>Data Module*. A primeira providência é criar a conexão com o banco usando um *SQLConnection* da paleta *dbExpress*. Clique duas vezes no componente e logo em seguida no botão "+". Selecione a opção *Interbase*. Digite um nome para a conexão e clique em OK.

Retornando ao editor de propriedades, selecione a opção *Database* e indique o caminho e nome do banco de dados criado anteriormente. Um exemplo disso seria "localhost:C:\BancoDeDados\SYSPAGUE.FDB". Nos parâmetros *User_Name* e *Password* digite o usuário e senha do banco.

Clique em OK e marque como *False* a propriedade *LoginPrompt* do *SQLConnection*. Ainda no *Object Inspector* modifique o *Name* para "sqlConexao". Modifique a propriedade *Name* do DM para "dmdPrincipal" e salve-o como "uDmPrincipal.pas".

Como dito anteriormente, vamos criar três telas de cadastro, por isso devemos criar os componentes de acesso para

todas elas. Começaremos por Clientes, assim adicione no Data Module um *SQLQuery* ("sqlClientes"), um *DataSetProvider* ("dspClientes") e um *ClientDataSet* ("cdsClientes").

As configurações dos componentes são simples e os passos seguidos aqui podem ser repetidos para cada grupo de componentes de acesso. No *sqlClientes* modifique a propriedade *SQLConnection* escolhendo o *sqlConexao* e em sua propriedade *SQL* adicione o código da **Listagem 2** .

Em seguida clique com o botão direito no componente e escolha a opção *Fields Editor*. No editor, clique novamente com o direito e escolha *Add all fields*. Isso fará com que todos os campos sejam carregados na lista de campos do *sqlClientes*. No *dspClientes* apenas modifique a propriedade *DataSet* para *sqlClientes*. Agora modifique o *ProviderName* do *cdsClientes* informando *dspClientes*.

Para terminar nossa primeira configuração, repita os passos de carregamento dos campos descrito no *sqlClientes* para o *cdsClientes*, ou seja, teremos os campos carregados no *Fields Editor* tanto no *SQLQuery* quanto no *ClientDataSet*.

Repita esses passos para as tabelas de *Fornecedores* e *Contas* sendo que cada uma deverá conter o *Select* adequado. Confira na **Tabela 1** o resumo das configurações necessárias, com o nome do componente e o valor da propriedade *SQL*.

Por motivos de performance, ou seja, para manter a velocidade de carregamento dos dados do servidor para o cliente, nossas queries retornarão sem registros. Por isso, usamos uma técnica na cláusula *where* das tabelas de Clientes e Fornecedores.

Note que a cláusula *where* foi modificada para CNPJ = -1, assim nenhum registro será retornado, pois não existem empresas com o CNPJ com o valor -1. A consulta por sua vez será aberta sem registros evitando carregar centenas de clientes/fornecedores desnecessariamente.

Para que o usuário altere ou visualize determinado Cliente/Fornecedor ele será obrigado a utilizar-se da caixa de pesquisa que criaremos, trazendo o mínimo possível de registros. Um *DBGrid* na parte lateral receberá os registros e o usuário poderá escolher o cadastro que será editado, caso seja necessário.

Diferentemente das tabelas já citadas (Clientes e Fornecedores) a tabela de Contas Correntes dificilmente possuirá centenas de registros, por esse motivo não usaremos a técnica descrita anteriormente.

Nesse ponto do artigo, o Data Module deverá se parecer com a **Figura 3**.

Adicionamos somente os componentes necessários para a conexão com as três tabelas criadas até o momento, ao longo do mini-curso adicionaremos novos componentes ao Data Module.

Agora vamos codificar os eventos *AfterDelete*, *AfterPost* e *BeforePost* de cada *ClientDataSet*. Esses eventos terão o papel de chamar o *ApplyUpdates* e também de atualizar a data de alteração (DT_ALTERACAO) da tabela em questão. Para o evento *AfterPost* do *cdsClientes* inclua o seguinte código:

```
cdsClientes.ApplyUpdates(0);
```

Já no *AfterDelete* apenas associe-o ao *AfterPost* selecionando-o na lista de eventos do componente. Para o *BeforePost* digite o código a seguir:

```
cdsClientes.FieldByName('DT_ALTERACAO').
 AsDateTime := Now;
```

Repita esses passos para os demais *ClientDataSets*, tomando o cuidado de mudar o nome do objeto.

Pesquisas e carregamento de dados

Antes de partirmos para o *design* e programação das telas de cadastro, devemos nos preocupar com o carregamento dos dados, ou seja, como nossas janelas se comportarão e receberão os registros presentes no banco.

É importante criar funções e/ou procedimentos apropriados para cada consulta. Criaremos as funções *CarregarTodos*, *PesquisarCliente*, *PesquisarFornecedores* e *PesquisarContas*, que receberão parâmetros das telas de cadastro e montarão os *selects* necessários para cada situação.

Na seção *public* do Data Module declare as funções mencionadas anteriormente como mostrado na **Listagem 3**.

Pressione CTRL+SHIFT+C para que o

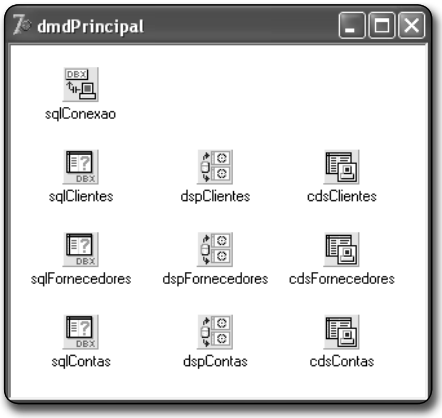


Figura 3. Data Module principal

Listagem 2. Consulta à tabela de Clientes

```
SELECT
  CNPJ, FANTASIA, RAZAO, IE, ENDERECO, BAIRRO,
  COMPLEMENTO, CIDADE, ESTADO, CEP,
  TELEFONE, FAX, DT_CADASTRO, STATUS, DT_ALTERACAO
FROM CLIENTES
WHERE CNPJ = -1 AND STATUS = 'A'
ORDER BY FANTASIA
```

Componente	Propriedade SQL
sqlFornecedores	SELECT CNPJ, FANTASIA, RAZAO, IE, ENDERECO, BAIRRO, COMPLEMENTO, CIDADE, ESTADO, CEP, TELEFONE, FAX, DT_CADASTRO, STATUS, DT_ALTERACAO FROM FORNECEDORES WHERE CNPJ = -1 AND STATUS = ëAí ORDER BY FANTASIA
sqlContas	SELECT BANCO, AGENCIA, CONTA, NOME_AGENCIA, NOME_CONTA, ENDERECO, BAIRRO, COMPLEMENTO, CIDADE, ESTADO, CEP, TELEFONE, FAX, DT_CADASTRO, CONTATO, STATUS, DT_ALTERACAO FROM CONTAS WHERE STATUS = ëAí ORDER BY BANCO, AGENCIA, CONTA

Tabela 1. Comandos SQL dos componentes de acesso a dados

Listagem 3. Funções para montar as instruções SQL

```
function CarregarTodos(ASqlQuery: TSqlQuery;
  ADataSet: TClientDataSet; ATabela, AOrderBy: string): Boolean;
function PesquisarCliente(ACampo, AValor: string; ATipoPesquisa: Integer): Boolean;
function PesquisarFornecedor(ACampo, AValor: string; ATipoPesquisa: Integer): Boolean;
function PesquisarContas(ACampo, AValor: string; ATipoPesquisa: Integer): Boolean;
```

Listagem 4. Função CarregarTodos

```
function TdmdPrincipal.CarregarTodos(ASqlQuery: TSqlQuery; ADataSet: TClientDataSet;
  ATabela, AOrderBy: string): Boolean;
var
  I, J, Conta: Integer;
  ListaCampos: TStrings;
begin
  with ASqlQuery, ADataSet do
  begin
    ADataSet.Close;
    ListaCampos := TStringList.Create;
    GetFieldNames(ListaCampos);
    Params.Clear;
    Sql.Clear;
    Sql.Add('SELECT ');
    for I := 0 to ListaCampos.Count - 1 do
      if I = ListaCampos.Count - 1 then
        Sql.Add('  ' + ListaCampos[I])
      else
        Sql.Add('  ' + ListaCampos[I] + ',');
    Sql.Add('FROM ' + UpperCase(ATabela));
    Sql.Add('WHERE STATUS = ''A'' ');
    { Monta o Order By }
    if Trim(AOrderBy) <> '' then
    begin
      Sql.Add('ORDER BY');
      Conta := 0;
      J := 1;
      for I := 1 to Length(AOrderBy) do
      begin
        Inc(Conta);
        if (AOrderBy[I] = ';') or (I = Length(AOrderBy)) then
        begin
          if I < Length(AOrderBy) then
            Sql.Add('  ' + Copy(AOrderBy, J, Conta - 1) + ',')
          else
            Sql.Add('  ' + Copy(AOrderBy, J, Conta));
          Conta := 0;
          J := I + 1;
        end;
      end;
    end;
    ListaCampos.Free;
  end;
  ADataSet.FetchParams;
  ADataSet.Open;
  Result := not ADataSet.IsEmpty;
end;
```

Listagem 5. Função PesquisarCliente

```
function TdmdPrincipal.PesquisarCliente(ACampo, AValor: string; ATipoPesquisa: Integer): Boolean;
var
  Operador: string;
begin
  case ATipoPesquisa of
    0: Operador := '=';
    1, 2: Operador := 'LIKE';
  end;
  with dmdPrincipal, sqlClientes, SQL do
  begin
    cdsClientes.Close;
    Clear;
    Params.Clear;
    Add('SELECT ');
    Add('  CNPJ, FANTASIA, RAZAO, IE, ENDERECO, '+'BAIRRO, COMPLEMENTO, CIDADE, ESTADO, ');
    Add('  CEP, TELEFONE, FAX, DT_CADASTRO, STATUS, '+'DT_ALTERACAO ');
    Add('FROM ');
    Add('  CLIENTES ');
    Add('WHERE ');
    { Monta a cláusula Where }
    Add('(UPPER' + ACampo + ') ' + Operador + ' :' + AValor + ')');
    cdsClientes.FetchParams;
    case ATipoPesquisa of
      0: cdsClientes.Params.ParamByName(AValor).AsString := UpperCase(AValor);
      1: cdsClientes.Params.ParamByName(AValor).AsString := UpperCase(AValor) + '%';
      2: cdsClientes.Params.ParamByName(AValor).AsString := '%' + UpperCase(AValor) + '%';
    end;
    Add('  AND STATUS = ''A'' ');
    Add('ORDER BY ');
    Add('  FANTASIA ');
    cdsClientes.Open;
    Result := not cdsClientes.IsEmpty;
  end;
end;
```

Delphi crie o cabeçalho de cada função na área *implementation* do código-fonte. Vá até o corpo da função *CarregarTodos* e digite o código da **Listagem 4**.

A função criada recebe como parâmetro os objetos *SQLQuery* e *ClientDataSet* que serão usados no carregamento dos dados. Recebe ainda a tabela e os campos usados na cláusula *order by*. Os campos da tabela são obtidos automaticamente pela função *GetFieldNames* do *ClientDataSet* recebido, e logo em seguida é adicionado à propriedade SQL do *SQLQuery*, também recebido via parâmetro.

Em seguida montamos a cláusula *from* com o nome da tabela e o modo de ordenação. Por fim, atualizamos e abrimos o *ClientDataSet*, devolvendo como resultado verdadeiro ou falso o que indica se existem dados no filtro realizado.

Agora vamos à função de pesquisa em Clientes, localizando o corpo da função *PesquisarCliente* e digitando o código da **Listagem 5**.

A função de pesquisa de clientes recebe como parâmetro o campo onde será realizada a busca, o tipo e o valor. Primeiro verificamos o tipo de busca. Caso seja EXATA usamos o operador "=", do contrário usamos o LIKE.

Após isso, fechamos o *cdsClientes*, limpamos seus parâmetros e atualizamos a propriedade SQL do *SQLClientes* com a novo *select* para pesquisa. Então, atualizamos novamente os parâmetros do *ClientDataset* e o abrimos retornando o resultado da busca.

As pesquisas de *Fornecedores* e *Contas* seguem o mesmo raciocínio mudando apenas os campos, objetos e o nome da tabela quando necessário (**Listagem 6**).

Essas pesquisas serão utilizadas pelas janelas de cada cadastro. As funções recebem como parâmetro: o campo onde é feita a busca (*ACampo*), o texto para a pesquisa (*AValor*) e o tipo de pesquisa (*ATipoPesquisa*) que pode ser: Pesquisa Exata, A partir do Início ou Em qualquer lugar. Dessa forma podemos localizar um registro digitando, por exemplo, a razão social completa, o início dela ou simplesmente pesquisando em qualquer lugar do campo RAZAO.

A função *CarregarTodos* tem o objetivo de carregar todos os registros da tabela, caso o usuário necessite de uma visualização

geral. Lembrando apenas de ter cuidado ao utilizar essa abordagem, já que podemos ter milhares de registros na tabela, o que implicaria em uma demora bastante grande no carregamento dos dados.

Outra informação importante é que estamos utilizando os mesmos componentes de acesso a dados para utilização nas pesquisas. Isso significa que a cada nova pesquisa realizada, a propriedade SQL dos *SQLQuerys* são sobrescritas e reabertas para filtrar os dados.

Uma dica seria a criação de componentes separados para cadastro e pesquisa, ou em alguns casos, até Data Modules diferentes.

Tela de cadastro padrão

Vamos desenvolver a parte visual de cada tela para que possamos visualizar como o exemplo ficará. Para isso, usaremos herança, então criaremos um formulário padrão e dele desenvolveremos as telas necessárias para este exemplo.

Para isso, crie um novo formulário usando a opção *File>New>Form*. Mude seu *Name* para "frmPadrao" e salve-o como "uPadrao.pas". As telas de cadastros devem parecer com a **Figura 4**.

À esquerda temos um painel de pesquisa que será usado para pesquisar os registros na base de dados e disponibilizarmos para consulta e/ou alteração. Já que não carregaremos todos os dados na abertura da janela, precisaremos de recursos para busca, ou seja, vamos evitar que o usuário carregue muitos registros em tela o que causaria lentidão e tráfego desnecessário na rede.

Ao final da tela temos três *Labels* e três *DBTexts*. Esses *DBTexts* estão vinculados aos campos DT_CADASTRO, STATUS e DT_ALTERACAO presente em cada tabela. Eles são responsáveis por informar a data que o registro foi criado, o estado atual dele e sua última alteração.

Listagem 6. Funções PesquisarFornecedores e PesquisarContas

```
function TdmdPrincipal.PesquisarFornecedor(
  ACampo, AValor: string;
  ATipoPesquisa: Integer): Boolean;
var
  Operador: string;
begin
  case ATipoPesquisa of
    0: Operador := '=';
    1, 2: Operador := 'LIKE';
  end;
  with dmdPrincipal, sqlFornecedores, SQL do
  begin
    cdsFornecedores.Close;
    Clear;
    Add('SELECT ');
    Add('  CNPJ, FANTASIA, RAZAO, IE, ENDERECO, '+
      'BAIRRO, COMPLEMENTO, CIDADE, ESTADO, ');
    Add('  CEP, TELEFONE, FAX, DT_CADASTRO, STATUS, '+
      'DT_ALTERACAO ');
    Add('FROM ');
    Add('  FORNECEDORES ');
    Add('WHERE ');
    { Monta a cláusula Where }
    Add('(UPPER' + ACampo + ') ' + Operador +
      ' :' + AValor + ')');
    cdsFornecedores.FetchParams;
    case ATipoPesquisa of
      0: cdsFornecedores.Params.ParamByName(
        AValor).AsString := UpperCase(AValor);
      1: cdsFornecedores.Params.ParamByName(
        AValor).AsString := UpperCase(AValor) + '%';
      2: cdsFornecedores.Params.ParamByName(
        AValor).AsString := '%' + UpperCase(AValor) +
        '%';
    end;
    Add('  AND STATUS = ''A'' ');
    Add('ORDER BY ');
    Add('  FANTASIA ');
    cdsFornecedores.Open;
    Result := not cdsFornecedores.IsEmpty;
  end;
end;

function TdmdPrincipal.PesquisarContas(
  ACampo, AValor: string;
  ATipoPesquisa: Integer): Boolean;
var
  Operador: string;
begin
  case ATipoPesquisa of
    0: Operador := '=';
    1, 2: Operador := 'LIKE';
  end;
  with dmdPrincipal, sqlContas, SQL do
  begin
    cdsContas.Close;
    Clear;
    Params.Clear;
    Add('SELECT');
    Add('  BANCO, AGENCIA, CONTA, NOME_AGENCIA, '+
      'NOME_CONTA, ENDERECO,');
    Add('  BAIRRO, COMPLEMENTO, CIDADE, ESTADO, CEP, '+
      'TELEFONE, FAX,');
    Add('  DT_CADASTRO, CONTATO, STATUS, '+
      'DT_ALTERACAO');
    Add('FROM');
    Add('  CONTAS');
    Add('WHERE ');
    { Monta a cláusula Where }
    Add('(UPPER(' + ACampo + ') ' + Operador +
      ' :' + AValor + ')');
    cdsContas.FetchParams;
    case ATipoPesquisa of
      0: cdsContas.Params.ParamByName(
        AValor).AsString := UpperCase(AValor);
      1: cdsContas.Params.ParamByName(
        AValor).AsString := UpperCase(AValor) + '%';
      2: cdsContas.Params.ParamByName(
        AValor).AsString := '%' + UpperCase(AValor) +
        '%';
    end;
    Add('  AND STATUS = ''A'' ');
    Add('ORDER BY');
    Add('  BANCO, AGENCIA, CONTA');
    cdsContas.Open;
    Result := not cdsContas.IsEmpty;
  end;
end;
```

ClubeDelphi PLUS www.devmedia.com.br/clubedelphi/portal.asp

Acesse agora o mesmo o portal do assinante ClubeDelphi e assista a uma vídeo aula de Everson Volaco que mostra como trabalhar com herança visual de formulários no Delphi.

www.devmedia.com.br/articles/viewcomp.asp?comp=1716

Esses campos serão atualizados automaticamente pelo sistema através dos eventos *AfterPost* e *AfterInsert* de cada *ClientDataSet* como vimos na fase de criação do Data Module.

Como configuração visual, temos ao topo uma *ToolBar* e dentro dela um *DBNavigator* ("nvtNavegador"), sete *ToolButtons* e alguns separadores. Temos também um *ImageList* com as imagens usadas nos botões mencionados.

Um detalhe bastante importante no formulário padrão é o *DataSource* que teremos que incluir. Ele será responsável por conectar-se ao *ClientDataSet* correspondente à tabela. Com ele também faremos uso de sua propriedade *DataSet* para que possamos chamar as funções de manipulação dos dados, tais como inserção, alteração, exclusão etc.

Vamos tratar agora de fazer a codificação dos botões que terão as funções básicas do cadastro: Incluir, Alterar, Salvar, Cancelar, Excluir, Visualizar Todos e Fechar.

> **Nota:** Modifique o *Name* de cada botão da barra como a seguir: "btnNovo", "btn Editar", "btnSalvar", "btnCancelar", "btn Excluir","btnVisualizarTodosí e "btnSair".

Antes de tudo crie uma nova *procedure* conforme o seguinte código:

```
procedure AtivarBotoes(AEdicao: Boolean);
```

Esse procedimento fará a ativação/desativação dos botões da barra de ferramentas, bem como outros elementos da tela. Pressione CTRL+SHIFT+C para que o Delphi crie o cabeçalho do procedimento. No corpo da procedure escreva o código da **Listagem 7**.

Os elementos que ainda não vimos até aqui e que constam em *AtivarBotoes* são:

• *nvtNavegador*: esse é o *DBNavigator* usado para navegar nos registros da tabela;

• *pnlPesquisa*: à esquerda temos um *Panel* onde encontram-se os campos para pesquisa.

Após isso, faremos a codificação de cada botão iniciando pelos botões *btnNovo*, *btnEditar*, *btnSalvar*, *btnCancelar* e *btn Excluir*. O código completo encontra-se na **Listagem 8**.

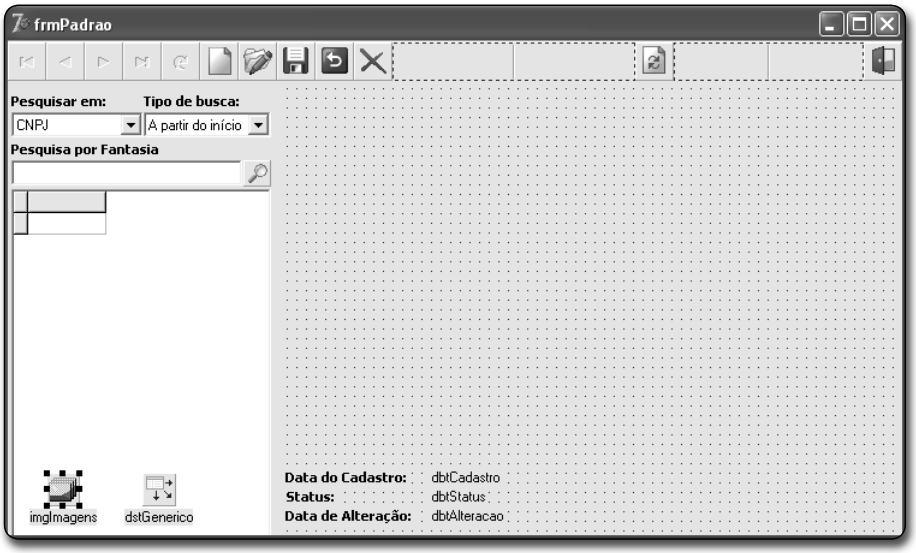


Figura 4. Exemplo de tela de cadastro

Note que a codificação é bastante simples dispensando explicações mais detalhadas a respeito, pois são apenas comandos comuns de manipulação de *DataSets*. O importante é notar que não estamos fazendo as chamadas aos métodos diretamente nos objetos presentes no Data Module principal.

Todos os métodos são utilizados através do *dstGenerico*, isso porque o formulário será herdado quando criarmos os cadastros. Os botões mais importantes nesse ponto são *btnVisualizarTodos* e *spbPesquisa* (no painel de pesquisa). Eles não recebem código porque serão personalizados e adaptados a cada situação.

Como mencionei, não permitiremos que todos os registros sejam carregados de início na tela. Por isso, o usuário será obrigado a fazer uma pesquisa no painel e localizar o registro desejado. Isso será feito utilizando-se os métodos de pesquisa criados no Data Module principal.

Mesmo usando esse conceito, caso o usuário queira forçar o carregamento de todos os registros da tabela, disponibilizamos o *btnVisualizarTodos*. Mostraremos uma mensagem de confirmação e caso o usuário aceite, o sistema faz um *select* no banco e traz para a aplicação quantos forem os registros na base. A montagem do painel de pesquisa consiste em inserir um *Panel* ("pnlPesquisa") alinhado *alLeft*.

Modifique também sua propriedade *BevelOuter* para *bvNone*. Dentro do *pnlPesquisa* adicione dois *ComboBoxes* com os nomes de "cbxPesquisa" e "cbxTipoBusca" respectivamente. Inclua também um *Edit* ("edtPesquisa") e um *SpeedButton* ("spbPesquisa"). Alinhado como *alBottom* coloque um *DBGrid*.

> **Nota:** Modifique também a propriedade *DataSource* do *nvtNavegador* e *DBGrid* ligando-os ao *dstGenerico*.

No primeiro *cbxPesquisa*, apenas configure para *csDropDownList* a propriedade *Style*. Os itens serão preenchidos usando o *GetFieldNames* do *DataSet* no evento *OnShow* do formulário, por isso digite o código da **Listagem 9** no evento mencionado.

No *cbxTipoBusca* inclua agora os itens "Exata", "A partir do início" e "Em qualquer lugar" e modifique a propriedade *Style* como *csDropDownList*. Com isso fi-

nalizamos a criação do nosso formulário padrão, agora basta adicionarmos ele ao repositório do Delphi.

Para isso, clique com o botão direito no formulário padrão e escolha a opção *Add to Repository*. Com a caixa de diálogo aberta, digite em *Title* o título do formulário padrão que salvaremos, nesse caso "FormPadrao".

Coloque uma descrição em *Description* e escolha a página *Forms* no item *Page* (**Figura 5**).

> **Nota**: Caso não queira adicionar o formulário explicitamente ao repositório, basta acessá-lo a partir da aba com o nome do projeto (*SysPague*) que aparece automaticamente no *repository*. Uma vantagem de adicionar o form dessa forma que fizemos é que ele fica disponível para outros projetos, além de receber os textos descritivos e ícone.

Cadastro de clientes

Observe que as tabelas de Clientes e Fornecedores são exatamente iguais, mudando apenas seu nome (uma alternativa seria criar uma tabela só, mas não foi a opção que escolhi).

Portanto a parte visual também deverá ser igual. Sendo assim vamos utilizar o recurso de repositório do Delphi para evitar redesenhar várias vezes a mesma tela.

Para utilizar o repositório clique em *File>New>Other*. Escolha a guia *Forms*, selecione o *FormPadrao*, marque a opção *Inherit* e confirme. Nesse ponto, criamos uma "cópia" perfeita do formulário padrão incluindo os eventos (herança).

O que devemos fazer agora é codificar os botões *spbPesquisa* e *btnVisualizarTodos*, além de inserir os *DBEdits* ligando-os a cada campo da tabela. Salve o formulário como "uClientes.pas" e mude seu *Name* para "frmClientes".

Nosso formulário é uma cópia herdada de padrão, por isso, será necessário configurar a propriedade *DataSet* do *dstGenerico* apontando-o para o *cdsClientes* no Data Module principal.

> **Nota:** Não esqueça de selecionar *File>Use unit* no Delphi e incluir a unit *dmdPrincipal* ao *uses* do formulário de Clientes.

Listagem 7. Código da procedure AtivarBotoes

```
procedure TfrmPadrao.AtivarBotoes(AEdicao: Boolean);
begin
  nvtNavegador.Enabled := not AEdicao;
  tbnNovo.Enabled := not AEdicao;
  tbnEditar.Enabled := not AEdicao;
  tbnExcluir.Enabled := not AEdicao;
  pnlPesquisa.Enabled := not AEdicao;
  tbnSair.Enabled := not AEdicao;
  tbnVisualizarTodos.Enabled := not AEdicao;
  tbnSalvar.Enabled := AEdicao;
  tbnCancelar.Enabled := AEdicao;
end;
```

Listagem 8. Código dos botões de manipulação do banco de dados

```
procedure TfrmPadrao.tbnNovoClick(Sender: TObject);
begin
  AtivarBotoes(True);
  if not dstGenerico.DataSet.Active then
    dstGenerico.DataSet.Open;
  dstGenerico.DataSet.Append;
  dstGenerico.DataSet.FieldByName('DT_CADASTRO').AsDateTime := Now;
  dstGenerico.DataSet.FieldByName('STATUS').AsString := 'A';
end;

procedure TfrmPadrao.tbnEditarClick(Sender: TObject);
begin
  if dstGenerico.DataSet.IsEmpty then
    Exit;
  AtivarBotoes(True);
  dstGenerico.DataSet.Edit;
end;

procedure TfrmPadrao.tbnSalvarClick(Sender: TObject);
begin
  AtivarBotoes(False);
  dstGenerico.DataSet.Post;
end;

procedure TfrmPadrao.tbnCancelarClick(Sender: TObject);
begin
  AtivarBotoes(False);
  dstGenerico.DataSet.Cancel;
end;

procedure TfrmPadrao.tbnExcluirClick(Sender: TObject);
begin
  if dstGenerico.DataSet.IsEmpty then
    Exit;
  if MessageDlg('Confirma exclusão do '+'registro atual?', mtConfirmation, [mbYes, mbNo],
    0) = mrYES then
    begin
      dstGenerico.DataSet.Edit;
      dstGenerico.DataSet.FieldByName('STATUS').AsString := 'I';
      dstGenerico.DataSet.Post;
      dstGenerico.DataSet.Close;
      dstGenerico.DataSet.Open;
    end;
end;
```

Listagem 9. Carregando ComboBox com os campos

```
procedure TfrmPadrao.FormShow(Sender: TObject);
begin
  cbxPesquisa.Items.Clear;
  dstGenerico.DataSet.GetFieldNames(cbxPesquisa.Items);
  cbxPesquisa.ItemIndex := 0;
  AtivarBotoes(False);
end;
```

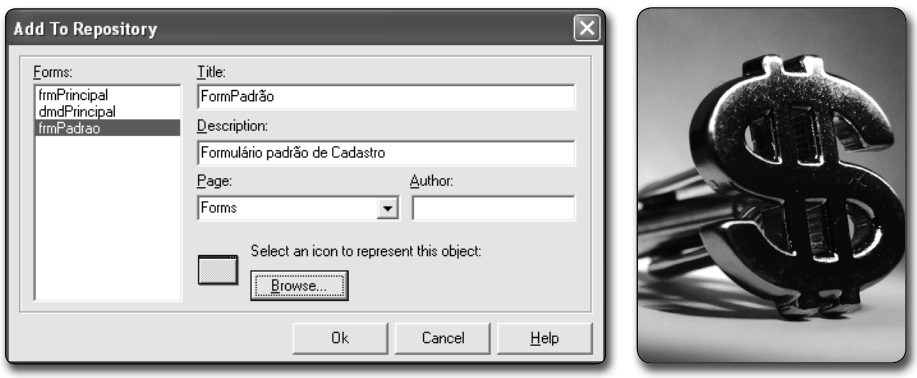


Figura 5. Adicionando o formulário ao Repositório

Após conectar o *dstGenerico* ao *cdsClientes*, vamos incluir o campo FANTASIA ao *DBGrid* de pesquisa. Para isso, clique duas vezes no *DBGrid* e em seguida clique no primeiro botão da caixa de diálogo que se abre.

Em *FieldName* escolha FANTASIA. Esse passo deverá ser repetido nas janelas de Fornecedores e Contas Correntes e é necessário, pois não adicionamos nenhum campo ao *DBGrid* quando desenhamos o formulário padrão, de onde herdamos.

Feito isso, o cadastro de Clientes estará quase completo. Vamos digitar o código dos botões *spbPesquisar* e *btnVisualizarTodos* (**Listagem 10**).

O *Pesquisar* consiste em chamar a função *PesquisarCliente* que criamos no Data Module principal e verificar se existem dados no resultado. Já o *CarregarTodos* faz um *select* no banco de dados e traz para a aplicação todos os registros encontrados na tabela passada como parâmetro.

A tela final deve ser semelhante à **Figura 6**, já com cada *DBEdit* apontando para o respectivo *DataSource* e *DataField*.

Listagem 10. Código dos botões btnPesquisar e btnVisualizarTodos

```
procedure TfrmClientes.spbPesquisaClick(Sender: TObject);
begin
  inherited;
  if (Trim(edtPesquisa.Text) = '') or (Length(edtPesquisa.Text) < 4) then
  begin
    MessageDlg('Digite ao menos 4 (quatro) letras '+
      'para realizar a pesquisa.', mtInformation, [mbOk], 0);
    edtPesquisa.SetFocus;
  end
  else
  begin
    with dmdPrincipal do
    begin
      if not PesquisarCliente(cbxPesquisa.Items[cbxPesquisa.ItemIndex], edtPesquisa.Text,
        cbxTipoBusca.ItemIndex) then
        MessageDlg('Nada selecionado para filtro.', mtInformation, [mbOk], 0);
    end;
  end;
end;

procedure TfrmClientes.tbnVisualizarTodosClick(Sender: TObject);
begin
  inherited;
  if MessageDlg('Essa operação poderá demorar '+'alguns minutos dependendo do volume de dados.' +
    #13 + 'Deseja prosseguir?', mtConfirmation, [mbYes, mbNo], 0) = mrYes then
    with dmdPrincipal do
      if not CarregarTodos(sqlClientes, cdsClientes, 'CLIENTES', 'FANTASIA') then
        MessageDlg('Nada selecionado para filtro.', mtInformation, [mbYes, mbNo], 0);
end;
```

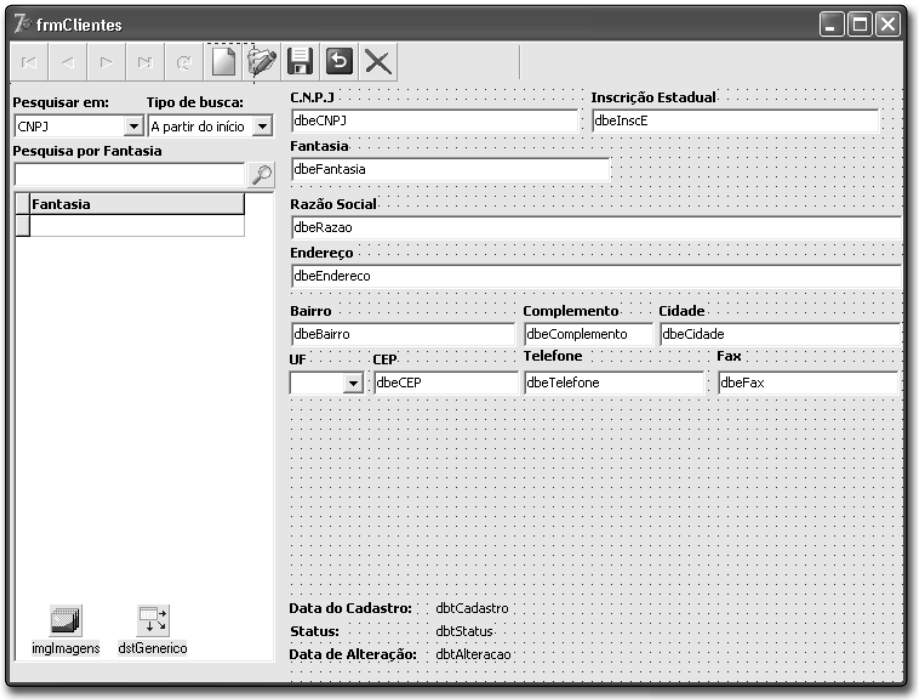


Figura 6. Cadastro de Clientes

Cadastro de Fornecedores

Para o cadastro de *Fornecedores* basta seguir todos os passos usados para a criação da tela de *Clientes*. É necessário apenas atentar aos detalhes cruciais dessa etapa:

- Ligar o *dstGenerico* ao *ClientDataSet* correto, nesse caso o *cdsFornecedores*;
- Fazer a chamada a *PesquisarFornecedor* no *spbPesquisa*;
- Configurar o código do *btnVisualizarTodos*.

Salve o formulário como "uFornecedores.pas" e troque seu *Name* para "frmFornecedores".

Cadastro de contas correntes

Nessa etapa, criaremos o último cadastro deste exemplo: *Contas Correntes*. Para poupar tempo vamos utilizar a tela padrão de cadastro criada em nosso repositório. Conforme mostrado anteriormente, adicione um novo formulário usando o mesmo método da criação dos formulários *Clientes* e *Fornecedores*, salve-o como "uContas.pas" e mude seu *Name* para "frmContas".

Os botões *btnVisualizarTodos e spbPesquisar* farão chamadas a *CarregarTodos* e *PesquisarConta*, respectivamente. Os códigos para ambos são os mesmos utilizados nas telas anteriores, atentando-se apenas as funções utilizadas e aos parâmetros a serem utilizados em cada situação.

No *DBGrid* de pesquisa, adicione apenas o campo NOME_CONTA. Estão faltando apenas os *DBEdits* ligados aos campos da tabela em questão. Basta ligá-los utilizando as propriedades *DataSource* e *DataField* como já estamos acostumados a fazer.

O *layout* de tela sugerido para este cadastro pode ser encontrado na **Figura 7**.

Pronto, com isso concluímos a criação de todos os cadastros. Aproveito pra fazer um comentário em relação ao *Excluir* e sua relação com o campo *Status* criado na tabela. Se observar, no evento *OnClick* do *btnExcluir* perceberá que não excluímos efetivamente o registro e sim alteramos o seu *Status* para *"I"* de inativo.

Dessa forma mantemos o registro sempre na tabela, de maneira que podemos resgatá-lo caso seja necessário. Para evitar

que o mesmo seja visualizado em telas de cadastro ou mesmo em relatórios não podemos esquecer de incluir na cláusula *where* da consulta, a verificação *Status = 'A'*.

É isso que fazemos em todas as consultas realizadas neste exemplo.

Concluindo o exemplo

Chegamos ao final do exemplo e precisamos agora apenas de alguns retoques para finalizar todas as etapas. Pressione CTRL+SHIFT+F11 ou acesse o menu *Project>Options*. Entre na guia *Forms* e selecione todos os formulários, com exceção do Data Module e da tela principal, e passe-os para a lista *Available forms*, ou seja, retire-os da lista de criação automática.

Ainda nessa tela, mova o Data Module para cima de maneira que seja o primeiro a carregar junto com o executável. Na aba *Application*, em *Title*, digite "SysPague – Contas a Pagar e Cobrança". Escolha um ícone se preferir e confirme.

Para finalizar, na tela principal, faça a chamada para cada tela de cadastro nos botões e menus correspondentes. O código completo para a chamada de cada item encontra-se na **Listagem 11**.

Conclusão

Nessa primeira etapa do mini-curso sobre Sistema de Contas a Pagar e Cobrança, criamos as três primeiras telas de cadastro necessárias para a manipulação de todos os dados do sistema. Nas próximas etapas criaremos os demais cadastros e entraremos nas fases de manipulação dos dados. ●

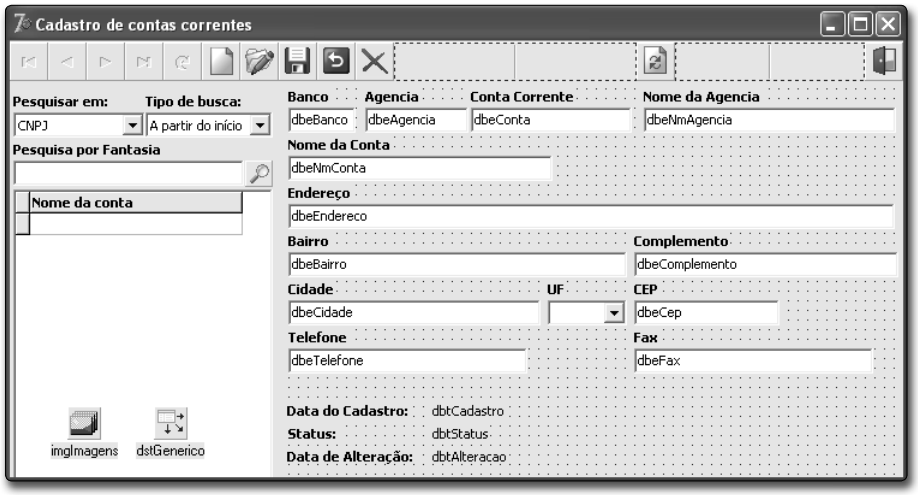


Figura 7. Exemplo de tela de Contas Correntes

Listagem 11. Chamada para as telas de cadastro

```
uses uClientes, uFornecedores, uContas;
...
procedure TfrmPrincipal.tbnClientesClick(Sender: TObject);
begin
  frmClientes := TfrmClientes.Create(Self);
  try
    frmClientes.ShowModal;
  finally
    frmClientes.Free;
  end;
end;

procedure TfrmPrincipal.tbnContasClick(Sender: TObject);
begin
  frmContas := TfrmContas.Create(Self);
  try
    frmContas.ShowModal;
  finally
    frmContas.Free;
  end;
end;

procedure TfrmPrincipal.tbnFornecedoresClick(Sender: TObject);
begin
  frmFornecedores := TfrmFornecedores.Create(Self);
  try
    frmFornecedores.ShowModal;
  finally
    frmFornecedores.Free;
  end;
end;
```

+ de 80.000 membros cadastrados
+ de 15.000 exemplos com fontes
+ de 900 apostilas
+ de 4.000 dicas
Fórum Delphi
Artigos
TOTALMENTE GRÁTIS
www.delphi.eti.br
Um dos maiores sites de apoio a desenvolvedores Delphi do Brasil!!!

Impressão em ECF

Como imprimir em impressoras fiscais

Adriano Santos
(artes@doiscliques.com)
é desenvolvedor Delphi desde 1998. Professor e programador PHP. Bacharel em Comunicação Social pela Universidade Cruzeiro do Sul, SP. É colunista e membro da Comissão Editorial da revista ClubeDelphi.

Neste artigo aprenderemos e entenderemos melhor como funciona um ECF (*Emissor de Cupom Fiscal*). Vamos ainda desenvolver um pequeno programa capaz de imprimir em impressora fiscal.

Para iniciarmos nosso projeto, precisamos primeiramente entender o que exatamente é um ECF e como funciona. Para isso vamos recorrer ao CONVÊNIO ECF 01/98. Nele estão dispostas a regras básicas para emissão do cupom fiscal. Conforme mencionado no convênio:

"Os estabelecimentos que exerçam a atividade de venda ou revenda de mercadorias ou bens, ou de prestação de serviços em que o adquirente ou tomador seja pessoa física ou jurídica não contribuinte do imposto estadual, estão obrigados ao uso de equipamento emissor de Cupom Fiscal - ECF."

Isso significa que todo e qualquer estabelecimento que efetue qualquer tipo de venda ou revenda ao consumidor com as características citadas, são obrigados a adquirir e se enquadrar com o esquema de emissão de cupom fiscal.

Os cupons fiscais são emitidos através do ECF, uma impressora fiscal de natureza matricial ou térmica. Essas impressoras coletam as vendas diárias emitindo ao final do dia a *Redução Z*, para fechamento ou troca de operador, e a *Leitura X* para início do período de vendas. Tanto a *Redução Z* quanto a *Leitura X* devem ser mantidas próximas ao ECF para fins de fiscalização quando solicitadas.

O ECF deve ser conectado a um PC ou PDV (*Ponto de Vendas*) compacto. Como mencionado, a prática é utilizada para o desenvolvimento de sistemas de vendas ao consumidor. Normalmente, utilizamos impressoras fiscais em lojas de conveniência, lanchonetes, bares e uma infinidade de outros estabelecimentos comerciais.

A interação entre o sistema e a impressora fiscal, é feita através das portas seriais, ou COM, por meio de bibliotecas

de link dinâmico, DLLs. Existem diversos fabricantes de impressoras fiscais, entre eles a Bematech (*www.bematech.com.br*).

Alguns tipos de contribuintes ficam isentos da utilização dos ECFís tais como empresas com faturamento inferior a R$ 120.000,00 anuais, empresas que não utilizem sistemas de processamento de dados ficam isentas de utilizar o ECF, estabelecimentos de ensino (Portaria 91/02) e corretoras.

Para uma listagem completa consulte o link *wiki.stoq.com.br/wiki/ECF_FAQs* que contém uma série de informações necessárias para o uso ou não do ECF. Uma série de regras são estabelecidas para a utilização correta do equipamento tais como:

- Somente é aceito o cancelamento de um cupom fiscal se o mesmo foi imediatamente cancelado após sua emissão;
- O cupom fiscal não deve ser emitido em caso de operações ou prestações com empresas contribuintes de ICMS;
- O cupom fiscal deve conter número do item, descrição, valor, código do produto vendido, quantidade, bem como valor pago pelo consumidor, troco e forma de pagamento.

Veja na **Figura 1** um exemplo de Cupom Fiscal impresso (emitido pelo emulador da impressora fiscal), com os produtos vendidos, total, valor pago e troco.

Emulando a impressora

Neste exemplo, vamos utilizar a impressora térmica fiscal MP-2100 TH FI da Bematech (emulador). A Bematech disponibiliza, gratuitamente, o emulador da impressora que pode ser utilizado para testar o equipamento e o sistema criado para esse propósito. Além do emulador, uma série de exemplos podem ser baixados no site do fabricante.

Entre no site *www.bematech.com.br* e acesse o menu superior *Suporte*. Logo em seguida clique no link *Downloads e Drivers* e após em *Impressoras Fiscais*. No *ComboBox* que aparece, selecione a sistema operacional Windows e aguarde o carregamento da página.

Localize *Emul2100 - VAREJO (versão 1.01)* e baixe o arquivo *Emul2100v101.zip*. Se preferir poderá baixar também o arquivo *FilmeEmul2100.zip* que contém um arquivo de vídeo no formato WMV com uma breve explicação sobre a utilização desse emulador.

Descompacte o arquivo e execute o programa de instalação (*Install Emul2100 v1.01.msi*). Após a instalação um grupo chamado *Bematech* aparecerá no seu menu *Iniciar>Programas* e nele o atalho para o emulador.

Abra o programa e clique com o botão direito, assim terá acesso a todos os controles do emulador tais como: ligar/desligar, porta serial, sensores etc. (**Figura 2**).

Além disso, será necessário efetuar o download da biblioteca *BemaFI32.dll* para que possamos desenvolver a aplicação exemplo, portanto, no mesmo link acessado, localize e baixe o arquivo *BemaFI32.zip*. Crie uma pasta onde salvaremos nosso exemplo e descompacte o arquivo nessa pasta.

Notará que, além da biblioteca, outros arquivos serão descompactados no diretório, tais como arquivos de help e configurações. Precisamos apenas dos arquivos *BemaFI32.dll* e *BemaFI32.ini*. Para usar o emulador, com nosso exemplo, é necessário alterar a chave *EmulMFD* de "0" para "1", por isso, abra o arquivo *BemaFI32.ini* e localize a chave informada. Altere para "1" e salve.

Criando o banco de dados de exemplo

Vamos criar duas tabelas simples em um banco de dados Firebird. Teremos a tabela PRODUTOS, com os produtos que serão vendidos e VENDAS, nossa tabela de pedidos. A estrutura de ambas pode ser vista no *script* da **Listagem 1**.

> **Nota:** Substitua *<Caminho>\ECF.fdb* pelo diretório do banco de dados e seu nome, ex: *C:\BancoDeDados\ECF.fdb*.

```
                CUPOM FISCAL
ITEM CÓDIGO DESCRIÇÃO QTD.UN.VL_UNIT( R$) ST VL_
ITEM( R$)
-----------------------------------------------
001 0000004458                  Boneco do Batman
        1 X 152,42        I1            152,42g
002 0000007786      Helicóptero Comandos em Ação
        1 X 127,42        I1            127,42g
003 0000007785                        Bate-Bate
        1 X 25,12         I1             25,12g
TOTAL  R$                             304,96
Dinheiro                              310,00
TROCO  R$                               5,04
-----------------------------------------------
Obrigado, volte sempre!
-----------------------------------------------
BEMATECH        MP-2100 TH FI        ECF-IF
VERSÃO:01.00.01                 ECF:001 LJ:0001
QQQQQQQQQQQQQUYYWI          13/07/2007 15:09:38
FAB:EMULADOR                              BR
```

Figura 1. Cupom Fiscal impresso

Figura 2. Emulador da impressora fiscal

Listagem 1. Criação do banco de dados e tabelas

```
SET SQL DIALECT 3;
SET NAMES WIN1252;
CREATE DATABASE 'localhost:<Caminho>\ECF.fdb'
USER 'SYSDBA'
PAGE_SIZE 8192
DEFAULT CHARACTER SET WIN1252;

CREATE TABLE PRODUTOS (
  ID          INTEGER NOT NULL,
  DESCRICAO   VARCHAR(30) NOT NULL,
  UNITARIO    NUMERIC(15,2) NOT NULL);

CREATE TABLE VENDAS (
  PEDIDO      INTEGER NOT NULL,
  ITEM        INTEGER NOT NULL,
  ID_PRODUTO  INTEGER NOT NULL,
  DESCRICAO   VARCHAR(30) NOT NULL,
  QTDE        INTEGER NOT NULL,
  UNITARIO    NUMERIC(15,2) NOT NULL);

ALTER TABLE PRODUTOS ADD CONSTRAINT PRODUTOS_PK
PRIMARY KEY (ID);
ALTER TABLE VENDAS ADD CONSTRAINT VENDAS_PK
PRIMARY KEY (PEDIDO, ITEM, ID_PRODUTO);
```

Data Module

Com o banco de dados criado, vamos criar a aplicação. Abra o Delphi, crie uma nova aplicação e salve a mesma. Vamos criar um Data Module e fazer as conexões necessárias para nosso exemplo. Clique em *File>New>Data Module*. Salve-o como "udmPrincipal.pas" e mude seu *Name* para "dmdPrincipal".

Vamos fazer uma conexão típica com dbExpress, para isso, adicione um *SQLConnection* ("sqlConexao") e dois *SQLDataSets* ("sdsProdutos" e "sdsVendas"). Clique duas vezes no *sqlConexao* e crie uma nova conexão apontando para o banco *ECF.fdb* que criamos anteriormente.

Marque como *False* a propriedade *LoginPrompt*. Em seguida modifique a propriedade *SQLConnection* de cada *SQLDataSet* apontando-os para o *sqlConexao*. Modifique a propriedade *SQL* do *sdsProdutos* incluindo o código a seguir:

```
SELECT ID, DESCRICAO, UNITARIO FROM PRODUTOS
ORDER BY ID
```

Em *sdsVendas* inclua o *select* a seguir:

```
SELECT ID, ITEM, ID_PRODUTO, DESCRICAO, QTDE,
UNITARIO
FROM VENDAS WHERE PEDIDO = -1
ORDER BY PEDIDO, ITEM, ID_PRODUTO
```

Após isso, clique duas vezes em cada *SQLDataSet* e adicione todos os campos da tabela usando a opção *Add all fields*. Inclua dois *DataSetProviders* ("dspProdutos" e "dspVendas"), ambos ligados aos seus respectivos *SQLDataSets* através da propriedade *DataSet*.

ClubeDelphi PLUS www.devmedia.com.br/clubedelphi/portal.asp
Acesse agora o portal do assinante ClubeDelphi e assista a uma vídeo aula de Guinther Pauli que explica como trabalhar com InternalCalc em ClientDataSets. Aula que faz parte do mini curso de ClientDataSet do autor.
www.devmedia.com.br/articles/viewcomp.asp?comp=5734

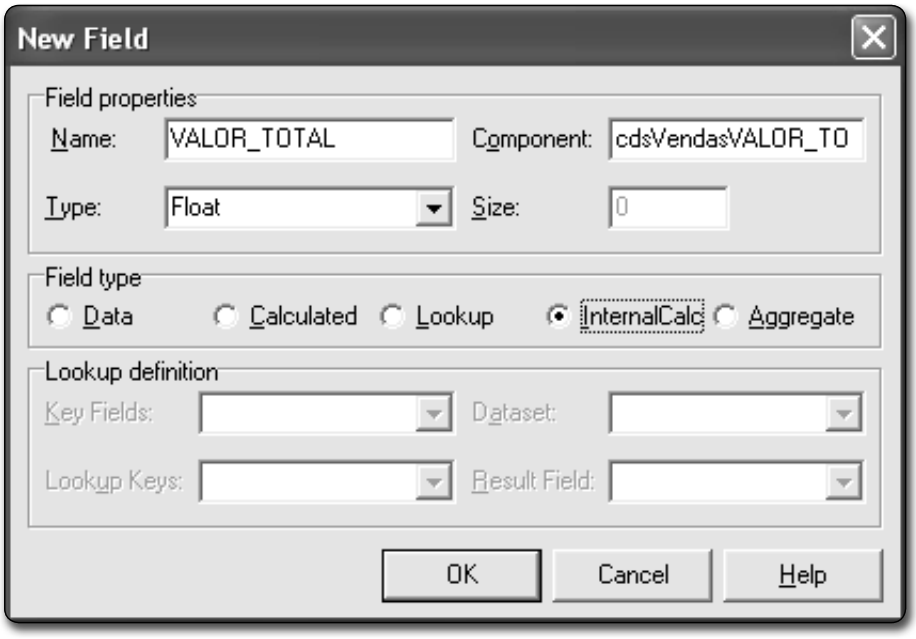


Figura 3. Criando um campo calculado do tipo InternalCalc

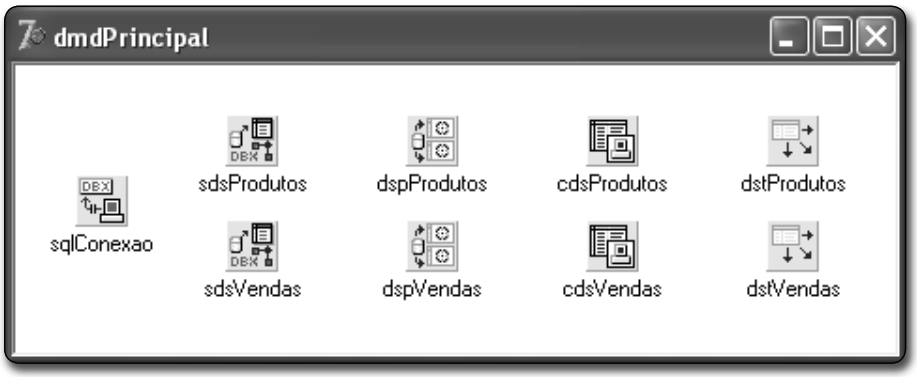


Figura 4. Data Module

Adicione dois *ClientDataSets* ("cdsProdutos" e "cdsVendas") e modifique a propriedade *ProviderName* de cada um, conectando-os a *dspProdutos* e *dspVendas*. Assim como nos *SQLDataSets*, inclua todos os campos da tabela.

Por último, adicione dois *DataSources* ("dstProdutos" e "dstVendas") e ligue-os aos *ClientDataSets* pela propriedade *DataSet*. Feito isso vamos inserir um campo do tipo *InternalCalc* nos *Fields* do *cdsVenda*.

Esse campo será usado para calcular automaticamente o valor de cada item adicionado ao pedido. Para isso, clique duas vezes no *cdsVendas* e no editor, clique com o botão direito e escolha *New Field*. Em *Name* coloque "VALOR_TOTAL", escolha o tipo *Float* e marque-o como *InternalCalc* (**Figura 3**).

Nesse momento devemos programar os eventos *AfterPost* e *AfertDelete* de cada *ClientDataSet* incluindo uma chamada ao *ApplyUpdates*. Isso facilita na hora de gravar os registros incluídos em cada operação dispensando o *ApplyUpdates* em tela.

Entre no evento *AfterPost* do *cdsProdutos* e inclua o código:

```
cdsProdutos.ApplyUpdates(0);
```

Após isso, ligue o evento *AfterDelete* ao *AfterPost* codificado anteriormente. Repita esses passos com o *cdsVendas* mudando somente o objeto que receberá o método, como:

```
cdsVendas.ApplyUpdates(0);
```

Precisamos agora programar os eventos *OnDestroy* e *OnCreate* do Data Module. A codificação é simples. Apenas abrimos e fechamos os *ClientDataSets* na criação e destruição do Data Module, respectivamente. Veja o código completo dos eventos na **Listagem 2**.

A última providência a se tomar é codificar o evento *OnCalcFields* do *cdsVendas*. Isso é necessário, pois o campo VALOR_TOTAL criado deverá receber o valor total de cada item, ou seja, o valor unitário multiplicado pela quantidade de produtos selecionada pelo operador. Para isso adicione o código a seguir, ao evento citado:

```
cdsVendas.FieldByName('VALOR_TOTAL').AsFloat :=
  cdsVendas.FieldByName('QTDE').AsFloat *
cdsVendas.FieldByName('UNITARIO').AsFloat;
```

O Data Module deve se parecer com a **Figura 4**.

Desenvolvendo a tela principal

Nossa aplicação terá a interface da **Figura 5.** Na parte superior da tela coloque um *GroupBox*. Adicione também um *MaskEdit* ("edtCPFCNPJ"). Configure a propriedade *EditMask* com a máscara "999.999.999-99;0;_" e mude o tamanho da fonte para "18".

Ao lado adicione um *SpeedButton* ("btnAbrirCupom") e à direita um *Label* ("lblMsgOperador"). Mude o tamanho da fonte para "18" e a cor para *clBlue*. Marque como *False* a propriedade *AutoSize* e *True* em *WordWrap*.

Aumente a largura e altura do *lblMsgOperador* de forma que se adapte melhor ao espaço deixado à direita do formulário. Inclua um novo *GroupBox* com o *Caption* "Dados do Produto e Pedido". Nele adicione um *DBGrid* ("dbgProdutos") e ligue-o ao *cdsProdutos* pela propriedade *DataSource*.

> Nota: Para que o formulário principal enxergue o Data Module, não esqueça de adicioná-lo à seção uses do formulário usando o menu File>Use unit.

Clique duas vezes no *DBGrid* e adicione todos os campos da tabela PRODUTOS. Configure as colunas, cores e fontes como desejar. À direita do *dbgProdutos*, adicione três novos botões que deverão se chamar "btnAdicionarItem", "btnCancelarItem" e "btnFecharVenda". Modifique os *Captions* para "Adicionar Produto", "Cancelar Item Anterior" e "Fechar e Totalizar".

Abaixo, insira um *Label* com o *Caption* "Quantidade" e ao lado um *SpinEdit* ("speQtde"). À direita, dois novos *Labels*, onde no primeiro apenas modifique seu *Caption* para "Pedido" e no segundo mude seu *Name* para "lblPedido".

Adicione também um novo *SpeedButton* ("speCancelarItemSelecionado") e mude seu *Caption* para "Cancelar Item Selecionado". Modifique sua largura se necessário. Insira um novo *DBGrid* ("dbgVendas") e aponte-o para *cdsVendas*. Adicione os campos da tabela e personalize como desejar. Para finalizar esse *GroupBox*, insira mais dois *Labels*, um com o *Caption* "SubTotal" e o outro com o *Name* para "lblSubTotal".

A área que acabamos de criar será responsável por todo o processo de criação do pedido, ou seja, será possível criar o pedido, incluir e excluir produtos, visualizar o Subtotal, escolher a quantidade de produtos a ser incluído e ainda iniciar o fechamento da venda.

A última etapa de desenvolvimento da parte visual do exemplo é criar o rodapé do formulário onde teremos as operações de escolha do tipo de pagamento, o valor pago pelo consumidor e a finalização do pedido.

Coloque um novo *GroupBox* e dentro adicione um *ComboBox* ("cbxFormPagto"). Em sua propriedade *Items* inclua os itens: "Dinheiro", "Cheque", "Vale" e "Duplicata". Em seguida insira um *Edit* ("edtValorPago") e um *BitBtn* ("btnPagar") e por fim, inclua um novo *BitBtn* maior com o nome de "btnFinalizarCupom".

Comunicação com a Impressora usando a DLL

Para codificar nosso exemplo precisamos antes declarar explicitamente todas as funções da impressora fiscal presente da biblioteca *BemaFI32.dll* para que possamos usar normalmente seus métodos.

Neste exemplo não usaremos todas as funções presentes na biblioteca, apenas o necessário. Por isso, abra o editor de

Listagem 2. Eventos OnDestroy e OnCreate do Data Module

```
procedure TdmdPrincipal.DataModuleDestroy(
  Sender: TObject);
begin
  cdsProdutos.Close;
  cdsVendas.Close;
end;

procedure TdmdPrincipal.DataModuleCreate(
  Sender: TObject);
begin
  cdsProdutos.Open;
  cdsVendas.Open;
end;
```

Figura 5. Interface de aplicação

códigos do Delphi e declare todas as funções da **Listagem 3**. Declare-as acima da palavra reservada *implementation*, ou seja, na área de *interface*.

Na **Tabela 1** temos as finalidades das funções descritas anteriormente, que são utilizadas para comunicar-se, através de vínculo estático, com as APIs da *BemaFI32.dll* que por sua vez envia os comandos ao ECF.

Funções auxiliares

Será necessário também criar algumas funções auxiliares pra evitar a digitação repetida de procedimentos importantes no projeto:

• *RetornarPedido*: Retorna o número do próximo pedido a ser criado;

• *InicializarVariaveis*: Inicia uma nova venda e ativa/desativa componentes visuais;

• *AtualizarMensagem*: Atualiza o *lblMsg Operador* informando mensagens ao operador do ECF.

Vá até a seção *public* do formulário e declare as funções e procedimentos como a seguir:

```
function RetornarPedido: Integer;
procedure InicializarVariaveis;
procedure AtualizarMensagem(AMensagem:
string);
```

Aproveite e declare também duas variáveis:

```
iPedido: Integer;
SubTotal: Double;
```

Pressione CTRL+SHIFT+C para que o Delphi crie o cabeçalho de todos os procedimentos. *RetornarPedido* cria em tempo de execução uma nova conexão e executa uma *select* na tabela de VENDAS usando *Max* para descobrir o maior número de pedido gerado. Logo após incrementá-lo, retorna o próximo pedido (uma opção seria usar um *Generator/Sequence* do Firebird).

Já *InicializarVariaveis* e *AtualizarMensagem* têm a responsabilidade de ativar e desativar botões, zerar variáveis e atualizar as mensagens enviadas ao operador por meio do *lblMsgOperador*, como dito anteriormente. Localize os procedimentos e codifique-os conforme a **Listagem 4,** onde temos todos os códigos necessários.

Abertura de cupom

Agora vamos iniciar a codificação de cada botão da interface. O exemplo consiste em abrir um novo cupom, função do *btnAbrirCupom*. Em seguida o operador deve escolher o item e a quantidade e adicionar/excluir ao pedido.

Por último é necessário informar o tipo de pagamento e o valor a ser pago fechando a venda. Após isso, o exemplo volta ao estado de *stand by*, ou seja, aguardando nova venda. Para iniciar o cupom digite o código da **Listagem 5** no *OnClick* do *btnAbrirCupom*.

O código chama o método que inicializa as variáveis e habilita os botões necessários para a venda. A abertura do cupom se dá pela chamada à *Bematech_FI_AbreCupom* informando, em seu único parâmetro, o CPF ou CNPJ do cliente.

Adicionando itens ao cupom e cancelando

Antes de iniciarmos a venda, devemos incluir os produtos ao pedido. Na fase de criação do banco de dados criamos apenas as tabelas, índices e o banco em si, porém não temos registros nas tabelas.

Entre no seu programa preferido para gerenciamento da base de dados (sugiro o IBExpert) e insira alguns produtos na tabela para que possamos testar a aplicação.

Listagem 3. Declaração das funções da biblioteca

```
{ Funções da BEMAFI32.dll }
function Bematech_FI_AbreCupom(
  CGC_CPF: string): Integer; stdCall;
  external 'BEMAFI32.DLL';
function Bematech_FI_VendeItem(Codigo: string;
  Descricao: string; Aliquota: string;
  TipoQuantidade: string; Quantidade: string;
  CasasDecimais: Integer; ValorUnitario: string;
  TipoDesconto: string; Desconto: string): Integer;
  stdCall; external 'BEMAFI32.DLL';
function Bematech_FI_IniciaFechamentoCupom(
  AcrescimoDesconto: string;
  TipoAcrescimoDesconto: string;
  ValorAcrescimoDesconto: string): Integer;
  stdCall; external 'BEMAFI32.DLL';
function Bematech_FI_TerminaFechamentoCupom(
  Mensagem: string): Integer; StdCall;
  external 'BEMAFI32.DLL';
function Bematech_FI_EfetuaFormaPagamento(
  FormaPagamento: string;
  ValorFormaPagamento: string): Integer; stdCall;
  external 'BEMAFI32.DLL';
function Bematech_FI_CancelaCupom: Integer; stdCall;
  external 'BEMAFI32.DLL';
function Bematech_FI_CancelaItemAnterior: Integer;
  stdCall; external 'BEMAFI32.DLL';
function Bematech_FI_CancelaItemGenerico(
  NumeroItem: string): Integer; stdCall;
  external 'BEMAFI32.DLL';
```

Função	Descrição
Bematech_FI_AbreCupom	Responsável por iniciar a venda.
Bematech_FI_VendeItem	Usada para enviar e imprimir o item vendido a impressora. A cada novo item vendido deverá obrigatoriamente ser enviado para impressão.
Bematech_FI_IniciaFechamentoCupom	Inicia o fechamento do cupom fiscal, ou seja, nesse ponto a venda o cupom fiscal é totalizado aguardando apenas a escolha da forma de pagamento e a entrada do valor para pagamento.
Bematech_FI_TerminaFechamentoCupom	Finaliza definitivamente a venda e prepara o ECF para uma nova venda.
Bematech_FI_EfetuaFormaPagamento	Envia para a impressora o valor e a forma de pagamento.
Bematech_FI_CancelaCupom	Cancela a venda.
Bematech_FI_CancelaItemAnterior	Cancela o último item enviado ao ECF.
Bematech_FI_CancelaItemGenerico	Cancela um item específico informado pelo parâmetro recebido. O parâmetro recebe o número do item a ser cancelado.

Tabela 1. Funções da BemaFI32.dll que serão usadas no exemplo

ClubeDelphi PLUS www.devmedia.com.br/clubedelphi/portal.asp

Acesse agora o mesmo o portal do assinante ClubeDelphi e assista a uma vídeo aula de Luciano Pimenta que mostra como baixar e instalar o IBExpert em sua versão gratuita.

www.devmedia.com.br/articles/viewcomp.asp?comp=3082

Os botões *btnAdicionarProduto* e *btnCancelarProduto* têm a função de incluir/excluir o item da tabela VENDAS bem como no cupom fiscal. Portanto, basta fazer chamada aos métodos *Append*, *Delete* ou *Post* do *cdsVendas* e os respectivos métodos da impressora, como podemos ver nas **Listagens 6** e **7**.

> **Nota:** Os parâmetros Alíquota e *TipoQuantidade* indicam a alíquota de imposto (ICMS ou ISS) e, como o próprio nome indica, tipo de quantidade. No primeiro podemos informar 4 ou 5 caracteres já com o valor da alíquota: ex 1800 ou 18,00 (com vírgula) ou ainda o índice da alíquota gravada na impressora que pode ser "I" ou "II". Se for o índice da alíquota deve ser 2 caracteres. Ex. (18,00 para o valor ou 05 para o índice). O *TipoQuantidade* pode ser informado como "I" para Inteira e "II" pra fracionária.

Note que adicionamos o item na tabela VENDAS, como também fizemos uma chamada à *Bemateh_FI_VendeItem* passando como parâmetro os dados do item vendido. Essa abordagem é obrigatória, ou seja, todo e qualquer item vendido deve ser enviado ao ECF para ser contabilizado e totalizado.

Note também que na ação do *Cancelar item anterior* chamamos outra função do ECF: *Bematech_FI_CancelaItemAnterior*, ou seja, o processo inverso a venda do item. Novamente o item em questão deve ser contabilizado, nesse caso cancelado pelo emissor de cupom fiscal.

Listagem 4. Código de RetornarPedido, InicializarVariaveis e AtualizarMensagem

```
function TfrmPrincipal.RetornarPedido: Integer;
var
  sqlTemporario: TSQLDataSet;
begin
  with dmdPrincipal do
  begin
    sqlTemporario := TSQLDataSet.Create(nil);
    sqlTemporario.SQLConnection := sqlConexao;
    with sqlTemporario do
    begin
      CommandText := 'SELECT MAX(PEDIDO) AS PEDIDO FROM VENDAS';
      Open;
      if FieldByName('PEDIDO').AsInteger = 0 then
        Result := 1
      else
        Result := FieldByName('PEDIDO').AsInteger + 1;
    end;
    sqlTemporario.Free;
  end;
end;



procedure TfrmPrincipal.InicializarVariaveis;
begin
  SubTotal := 0;
  iPedido := RetornarPedido;
  btnAbrirCupom.Enabled := False;
  btnAdicionarItem.Enabled := True;
  btnCancelarProduto.Enabled := False;
  btnFecharVenda.Enabled := False;
  btnPagar.Enabled := False;
  btnFecharCupom.Enabled := False;
  cbxFormasPgto.Enabled := False;
  cbxFormasPgto.ItemIndex := 0;
  edtValorPago.Enabled := False;
  edtValorPago.Text := FormatFloat('###,###,##0.00', 0);
  speQtde.Value := 1;
  speQtde.Enabled := True;
  lblPedido.Caption := FormatFloat('0000', iPedido);
  AtualizarMensagem('Iniciando Venda');
  lblSubTotal.Caption := FormatFloat('###,###,##0.00', SubTotal);
  edtValorPago.Text := FormatFloat('#####0.00', SubTotal);
  edtCPFCNPJ.Enabled := False;
  cbxFormasPgto.ItemIndex := 0;
end;

procedure TfrmPrincipal.AtualizarMensagem(AMensagem: string);
begin
  Application.ProcessMessages;
  lblMsgOperador.Caption := AMensagem;
  lblMsgOperador.Update;
end;
```

Listagem 5. Código de abertura do Cupom

```
procedure TfrmPrincipal.btnAbrirCupomClick(Sender: TObject);
begin
  InicializarVariaveis;
  Bematech_FI_AbreCupom(PChar(edtCPFCNPJ.Text));
end;
```

Listagem 6. Código do Adicionar Produto

```
procedure TfrmPrincipal.btnAdicionarItemClick(Sender: TObject);
var
  Item: Integer;
  Produto: string;
  Descricao: string;
  Qtde: string;
  Unitario: string;
begin
  with dmdPrincipal, cdsVendas do
  begin
    Last;
    Item := FieldByName('ITEM').AsInteger + 1;
    Append;
    FieldByName('PEDIDO').AsInteger := iPedido;
    FieldByName('ITEM').AsInteger := Item;
    FieldByName('ID_PRODUTO').AsInteger := cdsProdutos.FieldByName('ID').AsInteger;
    FieldByName('DESCRICAO').AsString := cdsProdutos.FieldByName('DESCRICAO').AsString;
    FieldByName('QTDE').AsInteger := speQtde.Value;
    FieldByName('UNITARIO').AsFloat := cdsProdutos.FieldByName('UNITARIO').AsFloat;
    Post;
    btnCancelarProduto.Enabled := True;
    btnFecharVenda.Enabled := True;
    btnCancelarItemEspecifico.Enabled := True;
    speQtde.Value := 1;
    Subtotal := Subtotal + cdsVendas.FieldByName('VALOR_TOTAL').AsFloat;
    lblSubtotal.Caption := FormatFloat('###.###,##0.00', SubTotal);
    edtValorPago.Text := FormatFloat('#####0.00', SubTotal);
    AtualizarMensagem('Item adicinoado');

    { Adiciona o item ao ECF, consequentemente, vendendo-o. Formata os dados necessários
      para a venda }
    Produto := FormatFloat('0000000000', FieldByName('ID_PRODUTO').AsInteger);
    Descricao := FieldByName('DESCRICAO').AsString;
    Qtde := IntToStr(FieldByName('QTDE').AsInteger);
    Unitario := FloatToStr(cdsVendas.FieldByName('UNITARIO').AsFloat);
    AtualizarMensagem(Descricao);

    { Função utilizada para realizar a venda de um item }
    Bematech_FI_VendeItem(PChar(Produto), PChar(Descricao), 'II', 'I', PChar(Qtde), 2,
      PChar(Unitario), '%', '0,00');
  end;
end;
```

Listagem 7. Código do Cancelar Produto

```
procedure TfrmPrincipal.btnCancelarProdutoClick(Sender: TObject);
begin
  with dmdPrincipal do
  begin
    Subtotal := Subtotal - cdsVendas.FieldByName('VALOR_TOTAL').AsFloat;
    lblSubtotal.Caption := FormatFloat('###.###,##0.00', SubTotal);
    edtValorPago.Text := FormatFloat('#####0.00', SubTotal);
    cdsVendas.Delete;
    btnCancelarProduto.Enabled := not cdsVendas.IsEmpty;
    btnCancelarItemEspecifico.Enabled := not cdsVendas.IsEmpty;
    AtualizarMensagem('Item cancelado');
    { Cancela também o item anterior do ECF }
    Bematech_FI_CancelaItemAnterior;
  end;
end;
```

Listagem 8. Código do Fechar e Totalizar

```
procedure TfrmPrincipal.btnFecharVendaClick(Sender: TObject);
var
  Produto: string; Descricao: string; Qtde: string;
  Unitario: string; AcrescimoDesconto: string;
  TipoAcrescimoDesconto: string;
  ValorAcrescimoDesconto: string;
begin
  with dmdPrincipal do
  begin
    with cdsVendas do
    begin
      AcrescimoDesconto := 'A';
      TipoAcrescimoDesconto := '%';
      ValorAcrescimoDesconto := '0,00';
      Bematech_FI_IniciaFechamentoCupom(PChar(AcrescimoDesconto),
        PChar(TipoAcrescimoDesconto), PChar(ValorAcrescimoDesconto));
      speQtde.Enabled := False;
      btnAdicionarItem.Enabled := False;
      btnCancelarProduto.Enabled := False;
      btnFecharVenda.Enabled := False;
      cbxFormasPgto.Enabled := True;
      edtValorPago.Enabled := True;
      btnPagar.Enabled := True;
    end;
  end;
end;
```

Fechamento do cupom

Adicionados os produtos, nosso operador deverá fechar / totalizar o pedido e aguardar a escolha da forma de pagamento e o pagamento em si. Nesse ponto devemos codificar o botão responsável pela primeira ação: *btnFecharVenda*.

Clicando no botão em questão, chamamos o *Bematech_FI_IniciaFechamentoCupom* que inicia o fechamento do cupom. Nesse ponto, a impressora já exibe o valor total da venda e aguarda a forma de pagamento e o valor pago (**Listagem 8**).

> **Nota:** Observe que cada botão clicado pelo operador, botões são ativados/desativados em seqüência para que o mesmo siga uma lógica na venda dos itens. Uma operação depende da outra.

Estamos chegando ao fim do desenvolvimento do sistema. Precisamos apenas codificar os botões *btnPagar* e *btnFecharCupom*. No *btnPagar* nosso principal procedimento é a chamada *Bematech_FI_EfetuaFormaPagamento*.

Ela recebe a forma de pagamento e o valor informado. Exemplo: Pagamento em Dinheiro no valor de R$ 200,00. Isso fará com que o ECF calcule o total do cupom e o troco que será dado se necessário (**Listagem 9**).

Agora basta chamarmos o *Bematech_FI_TerminaFechamentoCupom* presente na **Listagem 10** enviando a mensagem final ao ECF. Nesse ponto a impressora fiscal finaliza a venda e se prepara para nova emissão.

Um último método, bastante importante, é o código do *Cancelar item selecionado*. Esse botão pode ser utilizado caso o cliente deseje cancelar a compra de um item em específico. Nesse caso podemos fazer uma chamada ao *Bematech_FI_CancelaItemGenerico* passando como parâmetro o número do item a ser cancelado.

Em nosso exemplo foi passado o valor do campo ITEM da tabela VENDAS do registro (produto) selecionado. Dessa forma, basta o operador selecionar no *DBGrid* o item a ser cancelado e clicar no botão. O código de cancelamento específico é parecido com o código do *Cancelar item anterior*. Veja na **Listagem 11**.

Testando o ECF

Com isso finalizamos a codificação do exemplo. É importante lembrar que não colocamos codificações extras no fonte tais como: tratamentos de valores, tratamento de envio das funções à impressora, checagem de valores obrigatórios etc. É importante validar todas as entradas de dados para que não cause erros na emissão do cupom fiscal.

Para testar nosso exemplo abra primeiramente o emulador e clique com o botão direito do mouse e "ligue" a impressora. Siga as instruções do emulador para iniciar. Agora abra o sistema e efetue todos os procedimentos para emissão do cupom fiscal.

> **Nota:** Não foi incluído neste exemplo um botão para cancelamento do Cupom fiscal, porém essa operação é simples bastando chamar Bematech_FI_CancelaCupom.

Veja na **Figura 6** nosso sistema executando em conjunto com o emulador.

> **Nota:** Ao ligar o emulador, caso apareça alguma mensagem sobre erro de porta, não se preocupe. Verifique a mensagem do emulador, siga os passos e o emulador funcionará normalmente.

Conclusão

Neste artigo vimos como desenvolver facilmente um *Emissor de Cupom Fiscal*, usando como exemplo a biblioteca *BemaFI32.dll* e o emulador fornecido pelo fabricante. Para um maior aprofundamento dos fundamentos de ECF recomendo efetuar o download de exemplos no próprio site do fabricante, onde é possível encontrar diversas informações a respeito de impressão fiscal.

Até a próxima. ●

Links

Site do fabricante da Impressora Fiscal MP-2100 TH FI
www.bematech.com.br

FAQ com as principais perguntas e respostas sobre ECF
wiki.stoq.com.br/wiki/ECF_FAQs

Listagem 9. Código do botão Pagar

```
procedure TfrmPrincipal.btnPagarClick(Sender: TObject);
var
  ValorPago: string; FormaPgto: string;
begin
  if (StrToInt(FormatFloat(
    edtValorPago.Text, 0)) <> 0) then
  begin
    btnFecharCupom.Enabled := True;
    lblMsgOperador.Caption := '';
    Update;
    case cbxFormasPgto.ItemIndex of
      0: FormaPgto := 'Dinheiro';
      1: FormaPgto := 'Cheque';
      2: FormaPgto := 'Vale';
      3: FormaPgto := 'Duplicata';
    end;
    ValorPago := FormatFloat('###,###,##0.00', StrToFloat(edtValorPago.Text));
    Bematech_FI_EfetuaFormaPagamento(PChar(FormaPgto), PChar(ValorPago));
    cbxFormasPgto.Enabled := False;
    edtValorPago.Enabled := False;
    btnFecharCupom.Enabled := True;
  end
  else
  begin
    MessageDlg('Valor não pode ser igual a zero!', mtWarning, [mbOk], 0);
    edtValorPago.SetFocus;
  end;
end;
```

Listagem 10. Código de finalização

```
procedure TfrmPrincipal.btnFecharCupomClick(Sender: TObject);
var
  Mensagem: string;
begin
  try
    Screen.Cursor := crHourGlass;
    AtualizarMensagem('Finalizando venda');
    Mensagem := 'Obrigado, volte sempre!';
    AtualizarMensagem(Mensagem);
    Bematech_FI_TerminaFechamentoCupom(PChar(Mensagem));
    InicializarVariaveis;
    btnAbrirCupom.Enabled := True;
    edtCPFCNPJ.Enabled := True;
    edtCPFCNPJ.Text := '';
    edtCPFCNPJ.SetFocus;
    dmdPrincipal.cdsVendas.Close;
    dmdPrincipal.cdsVendas.Open;
  finally
    Screen.Cursor := crDefault;
  end;
end;
```

Listagem 11. Cancelamento de item específico

```
procedure TfrmPrincipal.btnCancelarItemEspecificoClick(Sender: TObject);
begin
  with dmdPrincipal do
  begin
    Subtotal := Subtotal - cdsVendas.FieldByName('VALOR_TOTAL').AsFloat;
    lblSubtotal.Caption := FormatFloat('###,###,##0.00', SubTotal);
    edtValorPago.Text := FormatFloat('#####0.00', SubTotal);
    { Primeiro cancela o item especifico do ECF }
    Bematech_FI_CancelaItemGenerico(IntToStr(cdsVendas.FieldByName('ITEM').AsInteger));
    { Agora apaga da pré-venda }
    cdsVendas.Delete;
    btnCancelarProduto.Enabled := not cdsVendas.IsEmpty;
    btnCancelarItemEspecifico.Enabled := not cdsVendas.IsEmpty;
    AtualizarMensagem('Item cancelado');
  end;
end;
```

Figura 6. Sistema em funcionamento

DevMedia
GROUP

O conteúdo ao seu alcance!

+ de **600 artigos** impressos por ano

+ de **1500 vídeo-aulas** de acesso imediato

+ de **24 cursos** online

+ de **130 vídeos inéditos** todos os meses

Se você busca **conteúdo** de qualidade e aprendizagem rápida.

Aqui é seu lugar!

Acesse e confira:

www.devmedia.com.br

Segurança de Sistemas

Controle de acesso e níveis de usuários

Maikel Marcelo Scheid
(maikelscheid@gmail.com)
é técnico em Informática com ênfase em Análise e Programação de Sistemas. Atua na área de Desenvolvimento de Softwares em Delphi para plataforma win32 e .NET com banco de dados Firebird e MS SQL há 3 anos.

Falar em segurança de sistemas, logo faz lembrar vários itens, que precisamos tomar os mais minuciosos cuidados, para que o sistema seja realmente seguro. Neste artigo veremos como definir de forma segura controles de acessos diferenciados para usuários de sistemas desenvolvidos em Delphi para a plataforma Win32.

Faremos uso do *User Control Package*, um pacote *open source* de componentes, que podem ser facilmente instalados nas versões posteriores ao Delphi 5, permitindo conexão aos principais padrões de acessos a dados e servidores de banco de dados compatíveis com Delphi. O acesso pode ser definido individualmente por Usuário e por Perfil de usuários, dando a opção de acompanhar uma lista de todos os usuários logados no sistema.

Iniciaremos com a instalação do componente, passando então à criação de uma aplicação Delphi com acesso a um banco de dados Firebird, onde aplicaremos o uso dos componentes do pacote.

Ao final do artigo, também veremos como gerar eventos no programa (*log*), podendo filtrar a consulta de *log* por usuário, data e nível.

O componente

Faremos o download do componente a partir do site mantido pelo desenvolvedor e colaboradores do projeto. Acessando o endereço *www.usercontrol.net* encontraremos um amplo canal de informações, dicas, fórum, entre outros, relacionados aos componentes do pacote.

Faça o download do pacote através da seção destinada no site. Para este artigo, faremos o download da versão *User Control 2.20 RC2*. Fique a vontade para fazer o download das outras versões disponíveis. Salve o arquivo compactado dentro da pasta *Lib* do seu Delphi (ou escolha outra pasta).

Após o download, descompacte o arquivo zipado, deixando os fontes na pasta *UserControl*.

Instalando o componente

Usaremos neste artigo para fazer a instalação, configuração e uso do componente, o Delphi 2006, mas você poderá usar livremente qualquer outra versão superior ao Delphi 5. A instalação do User Control é simples, no menu *File>Open* do Delphi, localize, dentro da pasta *UserControl/Packages*, o arquivo *pckMD5.dpk*.

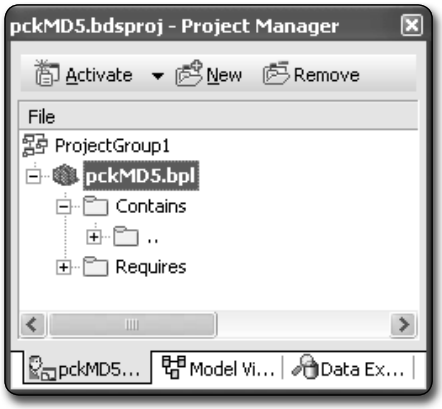


Figura 1. Instalação dos pacotes do componente

Abra o mesmo e no *Project Manager* clique sobre a BPL dando um *Build* (**Figura 1**).

Repita o processo com os pacotes *pckUCDataConnector.dpk*, *pckUserControl_RT.dpk* e depois *pckUserControl_DT.dpk*. Apenas no último, *pckUserControl_DT.dpk*, após o *Build*, com o botão direito sobre o pacote escolha a opção *Install*.

Uma mensagem de confirmação de sucesso de instalação do componente será apresentada. Na mensagem também serão exibidos os componentes que você acabou de instalar (**Figura 2**).

Depois de instalado o componente, confirme a mensagem e use o menu *File>Save All* para concretizar a instalação.

ClubeDelphi PLUS www.devmedia.com.br/clubedelphi/portal.asp

Acesse agora o mesmo o portal do assinante ClubeDelphi e assista a uma vídeo aula de Paulo Quicoli que mostra como instalar os componentes User Control no Delphi.

www.devmedia.com.br/articles/viewcomp.asp?comp=5675

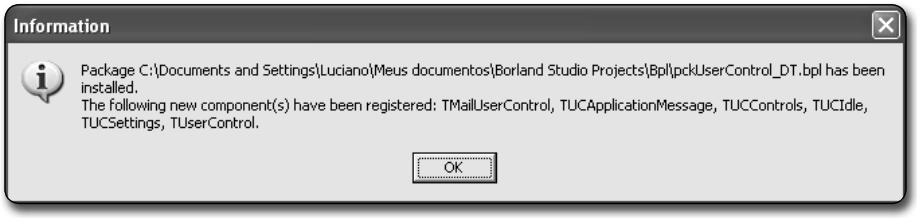


Figura 2. Mensagem de confirmação após instalação do componente

Instalando os conectores

Para que possamos realizar os controles de usuários e níveis de permissões, precisamos instalar os componentes responsáveis pela comunicação do User Control com os componentes de acesso a dados. Juntamente com os arquivos de instalação, dentro da pasta *Packages*, encontraremos uma pasta *Connectors*.

Para cada um dos principais tipos suportado pelo Delphi, teremos um conector a ser instalado. Tomaremos neste artigo como exemplo o acesso ao banco de dados Firebird, onde faremos a instalação do pacote *UCDBXConn*.

Se preferir, você poderá instalar todos os demais conectores, assim, estará habilitado para qualquer tipo de conexão. Para fazer a instalação é muito simples, semelhante a instalação do componente, basta abrir o pacote que será encontrada na pasta do pacote, *Build* e após *Install*. Novamente, você será notificado do sucesso da instalação do conector e poderá salvar a instalação.

> **Nota:** Lembre-se de adicionar à *Library Path* do Delphi (*Tools>Options*) o caminho da pasta com o fonte do componente, para que o mesmo possa fazer referência durante o uso.

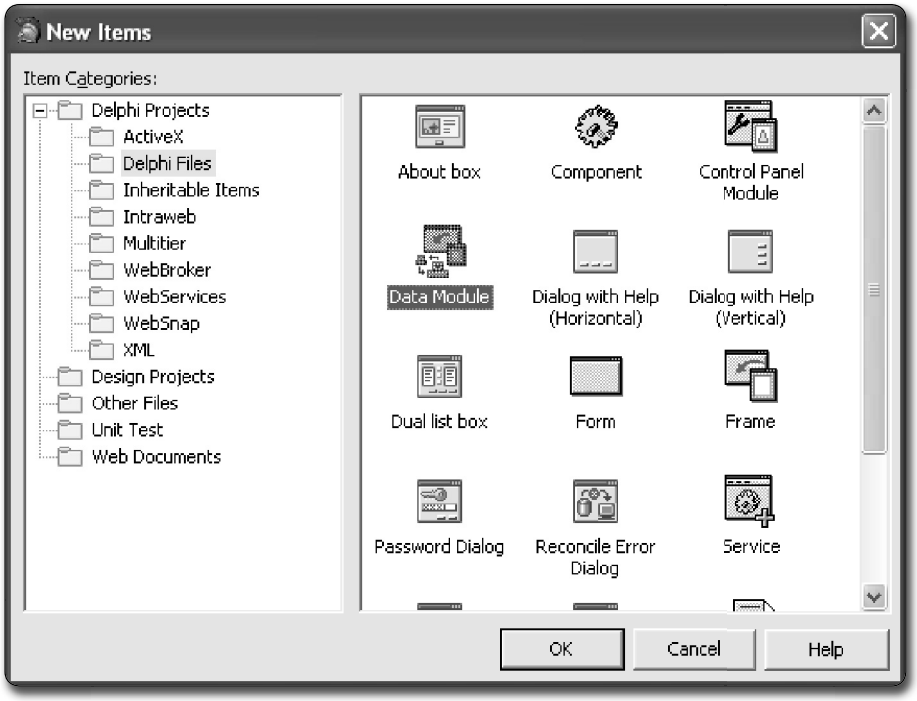


Figura 3. Criando o Data Module da aplicação

Iniciando uma aplicação

Criaremos agora uma aplicação no Delphi, onde utilizaremos um formulário principal e um Data Module para configuração do componente de acesso a dados. Iniciando uma nova aplicação *VCL Forms Application* (*File>New>VCL Forms Application*), criaremos uma pasta chamada "UCApp" onde salvaremos os arquivos da aplicação.

Salve o projeto com o nome de "UserControlApp.dpr" e o formulário como "UntUCApp.pas". Altere o *Caption* do formulário para "User Control" e o nome para "frmPrincipal". Através do menu *File>New>Other>Delphi Files*, selecione *Data Module* e confirme (**Figura 3**). Altere o nome do mesmo para "DMConexao" e salve como "UntDMConexao.pas".

Configurando a conexão

Adicione ao *DMconexao* um *SQLConnection*, altere seu nome para "Conexao" e com um duplo clique do mouse sobre o componente, abra o assistente para configuração do acesso ao banco de dados. Crie uma nova conexão, selecione o *driver Interbase* e em *Connection Name* dê o nome de "EMPLOYEE". Confirme e proceda com as configurações do acesso.

Utilizaremos o banco *EMPLOYEE.fdb* que acompanha a instalação do Firebird, e configuramos o *User_Name*, *Password* e *ServerCharSet* (**Figura 4**). Altere a propriedade *LoginPrompt* para *False* e ative a conexão.

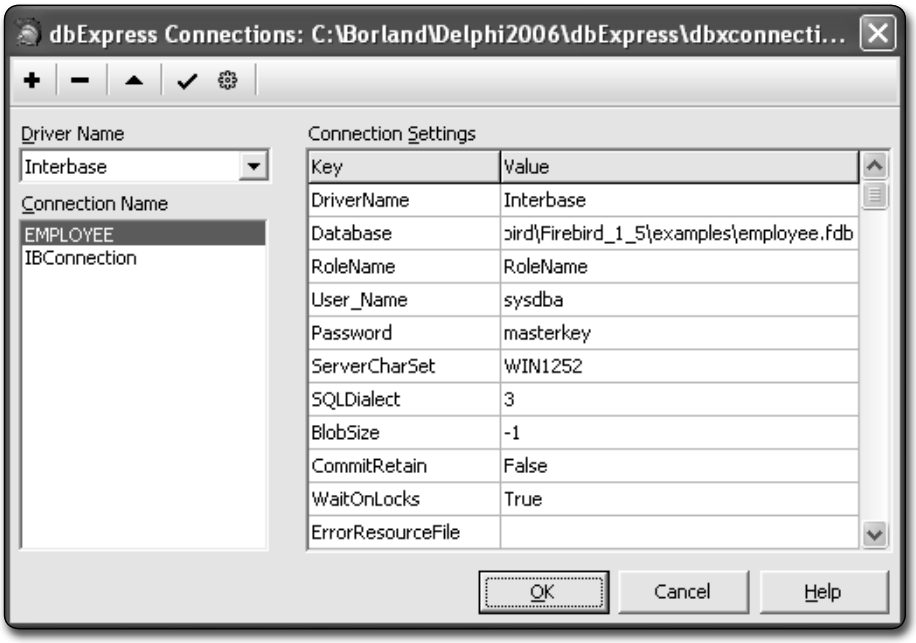


Figura 4. Configurando o acesso ao bando de dados Employee.fdb

Componente	Utilização do componente
UserControl	Principal componente do pacote. Todas as funções de controle serão centralizadas no mesmo. Ligue a propriedade Data Connector ao UCDBXConn que está no DMConexao. Veremos mais adiante as configurações mais detalhadas desse componente.
UCSettings	Permite a customização do UserControl. Podemos alterar textos e mensagens de telas de formulários do componente.
MailUserControls	Envia e-mails ao usuário, como por exemplo, para a opção Esqueci a senha. Ligue a propriedade MailUserControl do UserControl para esse componente.
UCIdle	Controla a inatividade do usuário. Poderá ser configurado um TimeOut para que o próprio sistema efetue logoff. Ligue a propriedade UserControl ao UserControl1.

Tabela 1. Componentes adicionados e função desempenhada por cada um

Arquivo	Usuários	Log
Logoff ("MLogoff") Sair ("MSair")	Alterar Senha ("MSenha") Cadastrar Usuários ("MUser") Perfis de Usuários ("MPerfil") Usuários Logados ("MLogados")	Log do Sistema ("MLog")

Tabela 2. Itens do menu

Propriedade	Menu da aplicação
User>MenuItem	MUser: Acessado pelo menu Cadastro de Usuários
UserPasswordChange>MenuItem	MSenha: Acessado pelo menu Alterar Senha
UserProfile>MenuItem	MPerfil: Acessado pelo menu Perfis de Usuários
UsersLogged>MenuItem	MLogados: Acessado pelo menu Usuários Logados
LogControl>MenuItem	MLog: Acessado pelo menu Log do Sistema
ApplicationID	UserControl: Nome da Aplicação

Tabela 3. Configurações das propriedades aos menus no componente

Nota: Uma prática segura é alterar a senha padrão do banco de dados após a instalação.

Utilizando o User Control

Arraste para o Data Module o *UCDBXConn* da paleta *UC Connectors* que acabamos de instalar. Faça a configuração da sua propriedade *Connection* ligando-a ao *SQLConnection*.

Volte para o *frmPrincipal* e solte os componentes mostrados na **Tabela 1**, da paleta *UC Main*.

Nota: Quando estiver no *frmPrincipal*, não esqueça de adicionar a referência ao Data Module, através do menu *File>Use Unit*, selecione o *UntDMConexao*. Não esqueça também de definir que o Data Module será criado antes que o formulário principal.

Em *Project>Options* arraste o Data Module para que fique posicionado acima ao *frmPrincipal*, isso garante que a conexão com o banco de dados já estará aberta quando o componente realizar uma consulta de usuário e senha na base.

Antes de executar a aplicação, precisamos definir os menus de acesso às funcionalidades do componente. Adicione ao *frmPrincipal* um *MainMenu*, e com um duplo clique crie uma estrutura semelhante à **Tabela 2** (os valores entre parênteses representam os nomes dos itens de menu).

No *UserControl*, definiremos as ligações das propriedades com cada um dos menus correspondentes. Configure cada uma das propriedades de acordo com a **Tabela 3**.

Para configurar o menu de efetuar *logoff*, adicione ao evento *OnClick* do *Logoff* o código a seguir:

```
procedure TfrmPrincipal.MLogoffClick(Sender:
TObject);
begin
  UserControl1.Logoff;
end;
```

Antes de compilarmos a aplicação, habilite a propriedade *AutoStart* do *UserControl* para *True*. Algumas configurações podem ser feitas em um editor, acessado ao dar um duplo clique no *UserControl*.

Testando a aplicação

Vamos executar a aplicação pela primeira vez, onde o componente criará automaticamente as tabelas por ele requeridas no banco de dados e mostrará uma tela com um usuário e senha padrão para proceder ao primeiro acesso (**Figura 5**).

Realizado o primeiro acesso ao sistema, acesse o menu *Usuários* e cadastre um novo usuário. No formulário de cadastro, criado automaticamente pelo componente (**Figura 6**), clique em *Adicionar* e faça o cadastro de seu próprio usuário.

Forneça todas as informações, exceto a opção *Perfil* que ainda está em branco e clique em *Gravar*, será habilitada uma opção onde deverá ser informada a senha para o novo usuário (essa é criptografada). Realizando um teste com o usuário criado, através do menu *Arquivo>Logoff*, acesse novamente o formulário de login e informe usuário e senha que acabou de cadastrar.

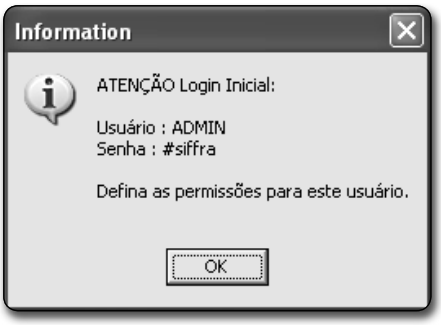


Figura 5. Usuário e senha padrão na primeira execução do sistema

Certamente você já notou que está com acesso a todos os menus do sistema e pensou em uma forma de como bloquear o acesso a determinado menu de acordo com cada usuário. Lembrando que durante as configurações criamos um menu chamado de *Perfis de Usuários*, vamos acessar o mesmo e criar um novo perfil, ou melhor, passaremos a criar dois perfis, um "ADMINISTRADOR" e outro chamado "USUARIOS".

Criados então os perfis, selecione o perfil USUARIOS e clique sobre o botão acessos. Apareceu um novo formulário, mas com nenhuma informação para definições de acesso. Não se preocupe você não cometeu nenhum erro, apenas não foi informado ao componente o que queremos controlar, nesse caso, serão os menus do sistema.

Feche a aplicação e voltando ao Delphi, no *UserControl* em sua propriedade *ControlRight>MainMenu* selecione o componente com a estrutura de menus (*MainMenu1*).

Compile e execute sua aplicação novamente, utilize para login o usuário ADMIN, certamente seu usuário não terá privilégios de acessos aos menus, e volte a acessar o menu *Perfis de Usuários*, selecione o perfil USUÁRIO e clique novamente em acessos.

Agora aparecendo todos os menus que configuramos anteriormente no componente, marque os mesmos de forma semelhante à **Figura 7** e confirme.

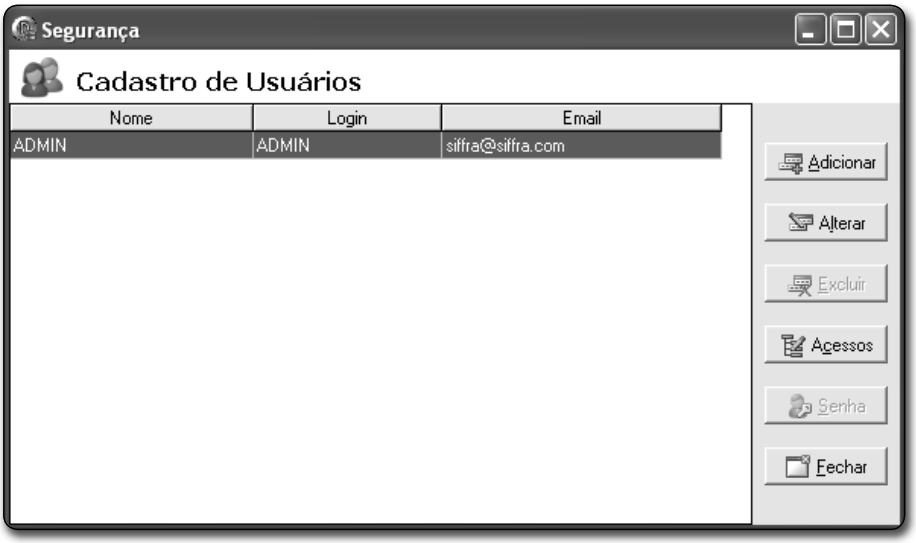


Figura 6. Tela para cadastro de um novo usuário do sistema

Após, selecione o perfil ADMINISTRADOR, defina acesso a todos os itens do menu e confirme. Acesse o menu *Cadastro de Usuário*, selecione o usuário ADM, e clique em *Alterar*. Defina-o como membro do perfil ADMINISTRADOR.

Repita o processo com o usuário que você cadastrou, determinando que o mesmo será membro do perfil USUARIOS. Efetue *logoff* e dessa vez utilize seu usuário para conectar ao sistema. Perceba que seus privilégios de acesso foram limitados, onde os menus continuam aparecendo, porém foram desabilitados.

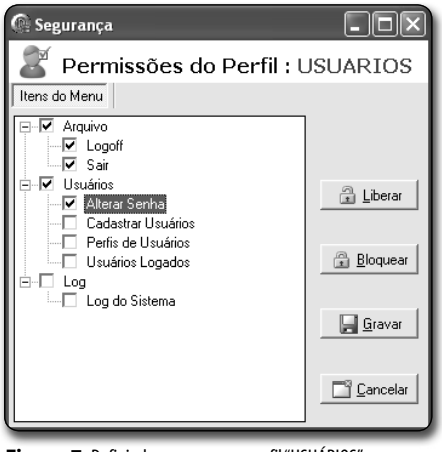


Figura 7. Definindo os acessos ao perfil "USUÁRIOS"

Acabamos de ver uma forma de controlar quem terá ou não acesso ao sistema, e aos menus que cada usuário terá acesso. Você poderá criar inúmeros perfis diferentes, e ainda, além de atribuir um usuário a um perfil, poderá definir acessos individuais a ele, na tela de cadastros.

Clique em *Acessos*, onde surgirá uma lista de definições individuais de acessos ao mesmo.

Funções de e-mail

Tendo adicionado o *MailUserControl* e configurado o mesmo ao *UserControl*, você possivelmente já deve ter notado a presença de um link *Esqueci a Senha* no formulário de login, também nas ações de cadastros e alterações de usuários, onde ao confirmar surgiu rapidamente uma tela de *Envio de Email*.

Todas essas opções são possíveis e já estão automaticamente habilitadas nas funcionalidades do componente, porém ainda não estão funcionando por não estarem devidamente configuradas. Então, mãos à obra.

Acessando as propriedades do *MailUserControl*, perceba que temos de informar ao mesmo os dados de uma conta de e-mail que serão utilizados para autenticação e o envio das informações. Precisamos informar ao componente o e-mail do remetente, o nome do mesmo e as informações do servidor de saída de e-mail (SMTP).

Informe o endereço do servidor, como por exemplo: "smtp.terra.com.br" ou simplesmente o número IP do mesmo. Informe também um usuário válido e a senha correspondente.

Simularemos o funcionamento do serviço de *Esqueci a Senha*, e definir a mensagem que será enviada para o e-mail do solicitante, juntamente com usuário e senha. Acessando a propriedade *EsqueceuSenha>Mensagem* e digite o seguinte texto:

```
Solicitação de Senha
Seu usuário é: :login
Sua Senha é: :senha
```

O texto digitado possui dois parâmetros, ou seja, login e senha que na hora da solicitação pelo usuário serão substituídos pelas devidas informações. Para testar, execute o programa, digite seu usuário e clique sobre o link *Esqueci a Senha*.

Você receberá em seu e-mail a senha cadastrada para seu usuário. Você poderá também alterar a mensagem que será enviada para o e-mail, bem como as informações que o e-mail carregará.

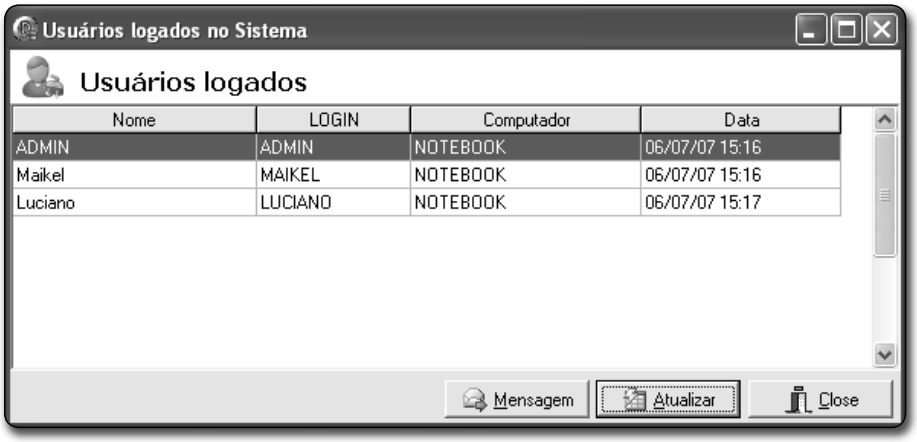
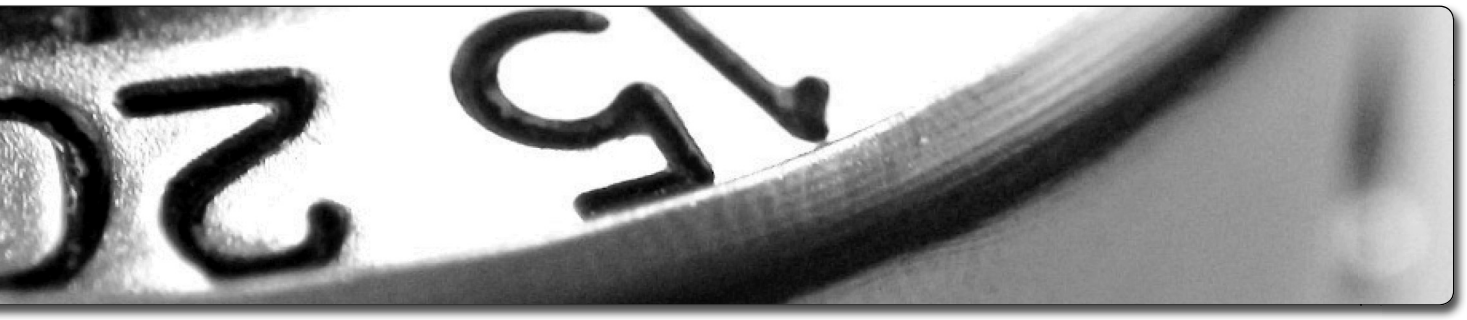


Figura 8. Usuários logados no sistema



Configure da mesma maneira o texto para as demais opções disponíveis no componente.

Timeout do sistema e lista de usuários logados

Muitos sistemas disponibilizam de recursos que controlam o tempo de inatividade dos usuários no sistema. No User Control, basta realizar uma configuração no *UCIdle*, alterando sua propriedade *Timeout* para o tempo que exista a seção para aquele usuário, sem o mesmo estar trabalhando no sistema (valor em *milissegundos*).

Em um exemplo de querermos atribuir um *timeout* de 20 minutos, bastaria atribuir à propriedade o valor "1200000", onde esgotado esse tempo, automaticamente o componente mostra a tela de login para que o usuário possa voltar a utilizar o sistema.

Se seu sistema está instalado em uma rede e você permitir acesso simultâneo de vários usuários, o User Control oferece uma funcionalidade para acompanhar todos os usuários conectados.

Já configurado anteriormente, basta você executar o sistema, e através do menu *Usuários Logados* obter a listagem dos mesmos que estão ativos (**Figura 8**).

Salvando os Logs das ações do usuário

Já que o assunto é segurança, porque não falar sobre *log*. É através do *log* de um sistema que o administrador poderá acompanhar todas as ações dos usuários no sistema. No *log* de um sistema podem ficar armazenadas informações muitos importantes em casos de *bugs* ou falhas de informações.

No *log*, qualquer ação do usuário poderá ser armazenada, juntamente com a data/hora e nome do usuário logado no momento da gravação do *log*, além é claro

da ação que o mesmo executou.

Adicione o seguinte código ao evento *OnLoginSuccess* do *UserControl*. Após, adicionado o código, execute o sistema e acesse o menu *Log*:

```
UserControl1.Log('Efetuou login no sistema', 0);
```

Acessando o *log* gerado pelo sistema, você já deve ter notado a presença de um registro que você fez login. Em qualquer formulário da aplicação você poderia acessar o componente e registrar o *log*, como por exemplo, em um formulário de cadastro, poderíamos criar um *log* quando o usuário insere um novo registro, altera, exclui, entre outras várias opções.

Um *log* pode ser armazenado com níveis diferentes de importância para o sistema, poderá ser armazenado como nível Baixo, de pouca importância para o administrador, como por exemplo, *login* e *logoff*.

Também como Normal, a exemplo de um usuário ter consultado algum relatório. Nível Alto, que geralmente é utilizado em alterações de registros e por final, um Crítico, correspondente a uma ação que o usuário fez e que poderá estar comprometendo informações do sistema, como por exemplo, a exclusão de um registro de estoque ou movimentações de caixa.

Esses níveis são definidos no momento da gravação do *log*, exemplo do código a seguir, onde informamos a ação que desejamos gravar e o nível da mesma:

```
UserControl1.Log('Efetuou login no sistema', 0);
{ Nível Baixo (0); Nível Normal (1); Nível
Alto (2);
  Nível Crítico (3) }
```

A tela de *log* permite ao usuário realizar várias opções de filtro, como por exemplo, consultar o *log* de um usuário específico, em uma determinada data, ou mesmo, pelo nível do *log* (**Figura 9**).

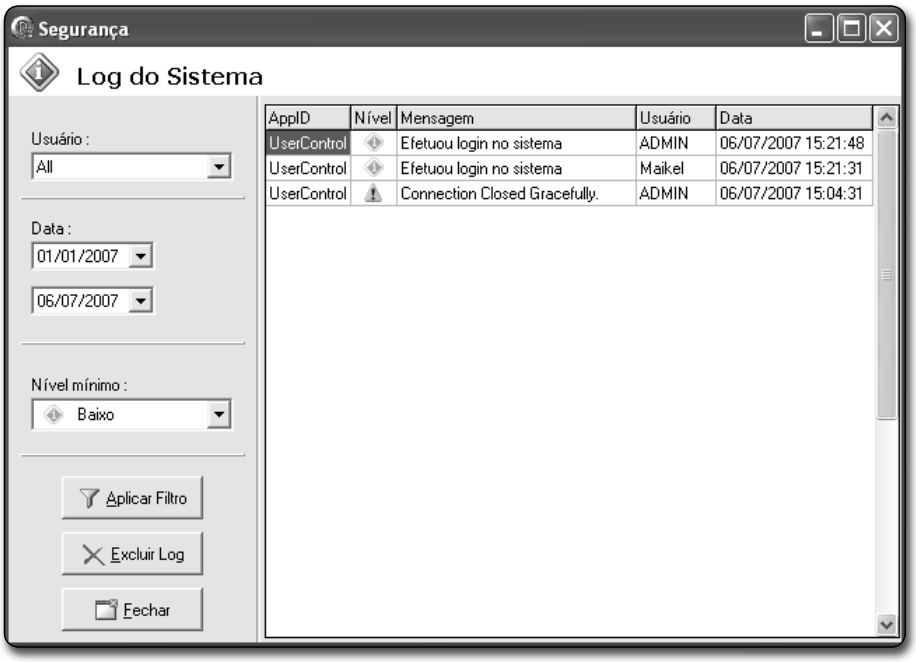


Figura 9. Tela de log do sistema

Outras configurações

No *UCSettings*, estão disponíveis inúmeras configurações para customização do User Control. Através desse componente é possível alterar todos os *Captions*, mensagens etc. Acessando as propriedades mudaremos a posição de todos os formulários do componente e também o *Caption* do login.

Alterando a propriedade *WindowsPosition* para *poScreenCenter* definimos que todos os formulários aparecerão ao centro da tela do computador. Para alterar o *Caption* do login, acessaremos a propriedade *Login>WindowsCaption* alterando-a para "User Control - Tela de Login". Execute novamente sua aplicação, veja a mudança (**Figura 10**).

Conclusão

Conforme comentado no início do artigo, vimos uma forma de evitar que pessoas indesejadas tenham acesso aos seus sistemas, limitando também as prioridades de acesso aos usuários, lembrando que as senhas ficam criptografadas no banco de dados.

Vimos também uma maneira fácil e segura de monitorar através do *log* tudo que acontece no sistema, tudo isso através de um processo de desenvolvimento muito ágil, seguro e com a utilização de poucas linhas de código. ●

Figura 10. Formulário de login com mudança no Caption

Links

Algumas informações foram obtidas do próprio site do componente
www.usercontrol.net

Realização: DevMedia GROUP

Últimos Ingressos!

31 de Agosto e 01 de Setembro
em São Paulo / SP

O Evento do desenvolvedor
Web e Wireless

Palestras 100% técnicas!
Garanta sua participação e conheça o que as tecnologias Java e .net trazem de melhor para este mercado!

Para mais informações acesse:
www.devmedia.com.br/eventos/webmobiletechweek

Confira abaixo a Grade de Palestras:

Salas de Java

1º dia - 31 de Agosto

	Sala 1	Sala 2	Sala 3
9:00-10:20	Interfaces ricas na Web com Ajax - Parte 1	Padrões de Projeto Java EE - Parte 1	WEB 2.0: Conceitos e Tecnologias
10:30-11:50	Interfaces ricas na Web com Ajax - Parte 2	Padrões de Projeto Java EE - Parte 2	Mercado de Jogos para Celulares
12:00-13:20	Almoço	Almoço	Almoço
13:30-14-50	Mini Curso 1 - Java ME Primeiros Passos - Parte 1	Struts2 - A Evolução do framework - Parte	Utilizando SVG (Scalable Vector Graphics) com Java ME
15:00-16:20	Mini Curso 1 - Java ME Primeiros Passos - Parte 2	Struts2 - Desenvolvendo uma aplicação com ajax e validação - Parte 1	Otimização de Aplicações Java ME
16:20-17:00	BREAK	BREAK	BREAK
17:00-18:20	Além do básico: explorando APIs sofisticadas com Java ME nas plataformas Nokia - Parte1	Struts2 - Desenvolvendo uma aplicação com ajax e validação Parte 2	iReport/JasperReport - Criando relatórios para web
18:30-19:50	Além do básico: explorando APIs sofisticadas com Java ME nas plataformas Nokia - Parte2	JavaFX: interfaces em Java de forma fácil e portável - Parte 1	Google Web Toolkit passo-a-passo - Parte 1
20:00-21:20	Interfaces ricas para a Web com Flex 2.0 e Java	JavaFX: interfaces em Java de forma fácil e portável - Parte 2	Google Web Toolkit passo-a-passo - Parte 2

2º dia - 01 de Setembro

	Sala 1	Sala 2	Sala 3
9:00-10:20	Implementando uma loja virtual em Java EE 5 com JPA e EJB3 - Parte 1	Desenvolvendo um Game em Java ME - Parte 1	Técnicas avançadas de stress-testing de aplicativos Web
10:30-11:50	Implementando uma loja virtual em Java EE 5 com JPA e EJB3 - Parte 2	Desenvolvendo um Game em Java ME - Parte 2	Desenvolvimento produtivo com Java Enterprise: JSF, JPA, AJAX e EJBs
12:00-13:20	Almoço	Almoço	Almoço
13:30-14-50	Implementando uma loja virtual em Java EE 5 utilizando JSF para a interface com o usuário.- Parte 1	Desenvolvendo um Game em Java ME - Parte 3	Java Wireless: O que nós precisamos saber sobre MIDP3?
15:00-16:20	Implementando uma loja virtual em Java EE 5 utilizando JSF para a interface com o usuário.- Parte 2	Introdução ao uso do bluetooth em J2ME	Mini Curso 2 – Conectividade em Java ME - Do básico ao avançado - Parte 1
16:20-17:00	BREAK	BREAK	BREAK
17:00-18:20	Desenvolvendo e implantando uma força de vendas em Palm - Parte 1	Desenvolvendo aplicações Web com o Spring Framework – Parte 1	Mini Curso 2 – Conectividade em Java ME - Do básico ao avançado - Parte 2
18:30-19:50	Desenvolvendo e implantando uma força de vendas em Palm - Parte 2	Desenvolvendo aplicações Web com o Spring Framework – Parte 2	Multimídia em J2ME RenatoFaria e eduardo peixoto

Salas de .NET

1º dia - 31 de Agosto

	Sala 1	Sala 2	Sala 3
9:00-10:20	Desenvolvimento para dispositivos móveis na plataforma .NET 2.0 - Parte 1	Linq - veja na prática os fundamentos deste projeto	Ciclo de vida de desenvolvimento com VSTS
10:30-11:50	Desenvolvimento para dispositivos móveis na plataforma .NET 2.0 - Parte 2	Mini-curso de C#	ASP.NET Caching
12:00-13:20	Almoço	Almoço	Almoço
13:30-14-50	Introdução ao ASP.NET AJAX	Estratégia de Soluções para Mobilidade & Casos de Sucesso	XAML
15:00-16:20	Aprenda na prática a criar um sistema de enquetes em ASP.NET - Parte 1	Acionamento de dispositivos eletro-mecanicos com Pocket PC	ASP.NET 2.0 na prática: Profiles e Membership
16:20-17:00	BREAK	BREAK	BREAK
17:00-18:20	Aprenda na prática a criar um sistema de enquetes em ASP.NET - Parte 2	Novidades no Windows Mobile 6	Criação de um portal com ASP.NET 2.0 e WebParts
18:30-19:50	Desenvolvendo programa para PocketPC com rede sem fio e WebServices - Parte 1	Introdução ao Silverlight	Desenvolvendo Gadgets Corporativos para Windows Vista
20:00-21:20	Desenvolvendo programa para PocketPC com rede sem fio e WebServices - Parte 2	Criando aplicações WAP com ASP.NET	.net 3.0 - Xaml Browser Applications

2º dia - 01 de Setembro

	Sala 1	Sala 2	Sala 3
9:00-10:20	Aplicações com ASP.NET 2.0 e AJAX - Parte 1	Otimizando aplicações em Windows Mobile 5.0	Aplicações Web para dispositivos móveis
10:30-11:50	Aplicações com ASP.NET 2.0 e AJAX - Parte 2	Construindo uma aplicação Mobile passo a passo com Visual Studio 2005	Linq para Compact Framework
12:00-13:20	Almoço	Almoço	Almoço
13:30-14-50	Construa uma aplicação 100% OO com ASP.NET - Parte 1	Envio e recebimento de dados no Pocket usando FTP	Silverlight na prática
15:00-16:20	Construa uma aplicação 100% OO com ASP.NET - Parte 2	Windows Mobile no Windows CE	WorkFlow Foundation no ASP.NET
16:20-17:00	BREAK	BREAK	BREAK
17:00-18:20	AJAX Control Toolkit	SQL Server 2005 Compact Edition - Parte 1	Desenvolvimento WEB: Uma Abordagem Utilizando OO e Técnicas de Teste de Software
18:30-19:50	Net Framework 3.0 - Conheça o WPF e surpreenda-se	SQL Server 2005 Compact Edition - Parte 2	Testes automatizados em aplicações ASP.NET

Patrocínio:

NOKIA Connecting People

Forum Nokia
www.forum.nokia.com

IBM developerWorks Live!

Caelum
Ensino e Soluções em Java
www.caelum.com.br

Globalcode

Apoio Cultural:

TEMPO REAL
LIVRARIA DE INFORMÁTICA E NEGÓCIOS

Apoio:

MOBILEZONE
Oficina da Net
aspbrasil
DFJUG
mcdbabrasil
TreoClub
<criarweb>
codigofonte.net
INFORMÁTICA
www.ramosdainformatica.com.br
NetCarreiras
VAGAS E CURRÍCULOS TI
Globo Bem

Cursos Online

Assinatura

ClubeDelphi PLUS

Mais conteúdo DELPHI por muito menos!

A Revista **ClubeDelphi** oferece para seus assinantes uma série de **Cursos Online** de alto padrão de qualidade . Conheça abaixo os cursos já disponíveis..

Curso em destaque

Aplicações Client/Server com dbExpress e Firebird

Confira neste curso online como criar uma aplicação client/server completa no Delphi 7, utilizando dbExpress e Firebird.
Aprenda também: Como trabalhar com um driver dbExpress específico e gratuito para o Firebird ; Como utilizar Orientação a Objetos, e técnicas como herança visual de formulários e relatórios ; saiba como colocar regras de negócios no banco de dados, usando Triggers e Stored Procedures; E crie relatórios com o Quick Report, Rave Reports e Report Builder, e muito mais.

Confira o plano de aula completo:
www.devmedia.com.br/clienteserver

A sua melhor opção de aprendizagem!
Assine a **ClubeDelphi** e comece já seu treinamento!
www.devmedia.com.br/assine

Outros cursos disponíveis: **www.devmedia.com.br/curso**

- Curso Online - Sistema SysCash
- Aplicações client/server com Windows Forms no Delphi 2006
- Criando uma Aplicação multi-camadas Completa com Delphi
- Aplicações WEB com IntraWeb e Delphi 7 (Delphi Win32)

DevMedia GROUP **Mais Informações: www.devmedia.com.br/central - Tel.: 21 2220-5375 / 2220-5435**